

COLD FUSION Developer's Journal

ColdFusionJournal.com

August 2000 Volume: 2 Issue: 8

Announcing...

Coming
November 12-15, 2000

**XML
DevCon
FALL
2000**

**JavaCON
2000**

September 24-27, 2000

Editorial

Making Money on Internet Time

Robert Diamond page 5

Foundations

Hors d' Oeuvres Anyone?

Hal Helms page 12

<BF> ON <CF>

Lock It or Lose It

Ben Forta page 16

CF Security

Security Made Simple

Kelly Brown page 38



**SYS-CON
MEDIA**



CFDJ Feature: WDDX & Data Sharing

How to use WDDX and Microsoft Excel for bulk data entry

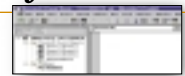


6

Tim Buntel

Live Monitoring of User Sessions

Know how many users are currently active in your apps



20

Christian Schneider

CF Techniques: Dynamic Goes Static

Create successful dynamic applications that can be turned into static pages with ColdFusion



24

Dustin Smith

Using Forms to Add or Change Data Part 2

A primer in dynamic page development for CF beginners

26

Ben Forta

CFDJ Feature: Introducing Smart Objects

Here's how to build extendable, reusable, object-based components using CFML



42

Benjamin Pate

Extending CFForm with Customized JavaScript Validation

Write your own functions to perform additional validation on your CFFORM tags

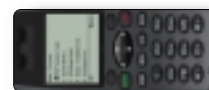


50

Selene Bainum

Developing Wireless Apps

Setting the stage for growing acceptance



56

Charles Arehart

abble commerce

www.ablecommerce.com

figleaf

www.figleaf.com

able commerce

www.ablecommerce.com

STEVEN D. DRUCKER, JIM ESTEN, BEN FORTA,
STEVE NELSON, RICHARD SCHULZE, PAUL UNDERWOOD

EDITOR-IN-CHIEF ROBERT DIAMOND
ART DIRECTOR JIM MORGAN
EXECUTIVE EDITOR M'LOU PINKHAM
MANAGING EDITOR CHERYL VAN SISE
ASSOCIATE EDITOR NANCY VALENTINE
PRODUCT REVIEW EDITOR TOM TAULI
TIPS & TECHNIQUES EDITOR MATT NEWBERRY

WRITERS IN THIS ISSUE

CHARLES AREHART, SELENE BAINUM, KELLY BROWN,
TIM BUNTEL, ROBERT DIAMOND, BEN FORTA, HAL HELMS,
BENJAMIN PATE, CHRISTIAN SCHNEIDER, DUSTIN SMITH

SUBSCRIPTIONS

SUBSCRIBE@SYS-CON.COM

FOR SUBSCRIPTIONS AND REQUESTS FOR BULK ORDERS,
PLEASE SEND YOUR LETTERS TO
SUBSCRIPTION DEPARTMENT.

SUBSCRIPTION HOTLINE 800 513-7111
COVER PRICE \$8.99/ISSUE
DOMESTIC \$79/YR. (12 ISSUES)
CANADA/MEXICO \$99/YR
OVERSEAS \$129/YR
BACK ISSUES \$12 EACH

PRESIDENT AND CEO UAT A. KIRCAALI
VICE PRESIDENT, PRODUCTION JIM MORGAN
VICE PRESIDENT, MARKETING CARMEN GONZALEZ
GROUP PUBLISHER JEREMY GEELAN
CHIEF FINANCIAL OFFICER ELLI HOROWITZ
ADVERTISING ACCOUNT MANAGER ROBYN FORMA
ADVERTISING ACCOUNT MANAGER MEGAN RING
ADVERTISING ASSISTANT CHRISTINE RUSSELL
ADVERTISING INTERN ALISON NOVICK
GRAPHIC DESIGNER ALEX BOTERO
GRAPHIC DESIGNER ABRAHAM ADDO
GRAPHIC DESIGN INTERN AARATHI VENKATARAMAN
WEBMASTER BRUNO Y. DECAUDIN
WEB DESIGNER STEPHEN KILMURRAY
CUSTOMER SERVICE ELLEN MOSKOWITZ
JDJ STORE.COM AMANDA MOSKOWITZ

EDITORIAL OFFICES

SYS-CON MEDIA, INC. 135 CHESTNUT RIDGE RD.,
MONTVALE, NJ 07645
TELEPHONE: 201 802-3000 **FAX:** 201 782-9600

COLDFUSION DEVELOPER'S JOURNAL (ISSN #1523-9101)
IS PUBLISHED MONTHLY (12 TIMES A YEAR)
FOR \$79 BY SYS-CON PUBLICATIONS, INC.,
135 CHESTNUT RIDGE RD., MONTVALE, NJ 07645

POSTMASTER

SEND ADDRESS CHANGES TO:

COLDFUSION DEVELOPER'S JOURNAL
SYS-CON MEDIA, INC.

135 CHESTNUT RIDGE RD., MONTVALE, NJ 07645

© COPYRIGHT

COPYRIGHT © 2000 BY SYS-CON MEDIA, INC.

ALL RIGHTS RESERVED. NO PART OF THIS PUBLICATION MAY BE
REPRODUCED OR TRANSMITTED IN ANY FORM OR BY ANY MEANS,
ELECTRONIC OR MECHANICAL, INCLUDING PHOTOCOPY OR ANY
INFORMATION STORAGE AND RETRIEVAL SYSTEM,
WITHOUT WRITTEN PERMISSION.

FOR PROMOTIONAL REPRINTS, CONTACT REPRINT COORDINATOR.

SYS-CON PUBLICATIONS, INC., RESERVES THE RIGHT TO REVISE,
REPRODUCE AND AUTHORIZE ITS READERS TO USE
THE ARTICLES SUBMITTED FOR PUBLICATION.

WORLDWIDE DISTRIBUTION

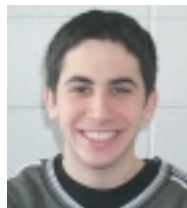
BY CURTIS CIRCULATION COMPANY 730 RIVER ROAD,
NEW MILFORD, NJ 07466-3048 PHONE: 201 634-7400

DISTRIBUTED IN USA

BY INTERNATIONAL PERIODICAL DISTRIBUTORS
674 VIA DE LA VALLE, SUITE 204, SOLANA BEACH, CA 92075
619 481-5928

ALL BRAND AND PRODUCT NAMES USED ON THESE PAGES
ARE TRADE NAMES, SERVICE MARKS OR TRADEMARKS
OF THEIR RESPECTIVE COMPANIES.

Making Money on Internet Time



BY ROBERT DIAMOND

Living on Internet time proves more and more interesting every day. Sometimes it seems the online world is moving so fast that were I to sleep late one day, I might wake up and find that I'd missed an entire revolution – or at the very least another “correction” to the stock market before which I ought to have sold!

When speaking about the market and about new technology, the word *innovation* always comes up – because that's the politically correct thing to tout. But what lies beneath the innovation? Money...very possibly lots of it.

Appearing on CNN's financial news and saying you're in it to get rich isn't well advised. The desire to have more, get more, make more and spend more – all while working less – is the new American dream. Everyone I know, from programmers to Web page designers to those doing data entry are doing more than just watching this incredible growth with fascination; they're all trying to get in on it – any way they can.

Hardly a day goes by, or a dinner or even a simple night out, without a friend pulling me aside. “What do you think of this?” “What the Internet needs is ...” Or best of all (and my personal favorite): “I've a great new idea that'll make us both rich.” Sure it will. They've come up with an idea, ranging from mundane to brilliant, have done a quick Web search to see if it already exists and have usually already gone through the final and seemingly most difficult stage of starting a new business – picking a virgin domain name. Network Solutions records a new registration about every five seconds – and while every domain name isn't taken, it often seems that every one that makes logical sense has been. (A word of advice on that subject: keep your eyes on international domain name resellers like .CC, they seem poised to catch on. Picking up a few key domains there surely couldn't hurt...)

Okay, back to the story. Our dot-com gazillionaire wannabes have now safely registered FantastikIdea12345.com (fantastic with the typo was all they could get their hands on) and with their big plan in mind...they're all ready to go.

What they need next is someone to make the site and that's usually where I fit into the picture. Having followed the world of ColdFusion with a tad of interest they're positive that no matter what site they want to create, ColdFusion can do it, that it can do it fast and probably without going over the budget (also known as their credit card limit). What's great about CF is that, in the majority of cases, they're right. While not *always* the best solution for a site, and while using it alone mightn't be the most scalable of ideas, CF almost always can be made to work. (On the topic of when ColdFusion isn't the right choice, there was a great article by Ben Forta a few issues ago that I highly recommend: “When Not to Use CF” [*CFDJ*, Vol. 2, issue 2, available in our online archives at www.sys-con.com/coldfusion/archives/0202/forta.1].)

After the site has launched comes the hardest part of the process and the one that no one has developed a formula or an exact science for as yet...getting traffic to the site. The greatest site is only as great as the people coming to it, and if they don't know it's there, then they certainly aren't coming to it. There are several grand options available for those with large amounts of funding, such as purchasing a \$100,000 ad on Yahoo!'s main page or buying an even more astronomically expensive national primetime television ad (SuperBowl, anyone?). For those on a shoe-string budget the choices are much slimmer – and potentially less effective. The goal of most marketing campaigns is to create word-of-mouth buzz and the best way to do that, big or small, is to make a good site: one that works reliably and provides visitors with value they can't obtain elsewhere. I'm a firm believer that if you build it well, people will come. As the early summer stock market shake-ups have shown, what is rewarded most is profits (big surprise). But how do you generate the profits? With a well-done site, of course. We here at *CFDJ* are dedicated to helping you create just that.



ABOUT THE
AUTHOR
*Robert Diamond is
editor-in-chief of
ColdFusion Developer's
Journal.*

Robert Diamond



WDDX & Data Sharing

How to use WDDX and Microsoft Excel for bulk data entry

BY TIM BUNTEL

WDDX exists to allow us to share data. With it your ColdFusion application can “talk” to someone else’s Perl script, and it in turn can talk to someone else’s Microsoft Word document. Quite a powerful concept, but I have to admit I haven’t had a lot of opportunities to use it.

Much of the focus on WDDX has centered on syndication. “Web syndication is the idea that the content and commerce assets of a corporate Web site can be exposed as services and data to other Web sites, allowing sites to *syndicate* their value to other Web sites” [Wddx.org]. Now don’t get me wrong. This is a very appealing idea and certainly could revolutionize many business relationships. Unfortunately, most of my clients are still trying to figure out how to best use their own data themselves, never mind sharing it with others! Accordingly, I’d like to take a moment to look into how WDDX can be used for more mundane data-sharing tasks.

I recently completed a large project in which users in many different organizations within a large company contributed to a

fairly complex database. A rich Web interface was developed to input records and was made as painless as possible considering the number of fields to be completed. Only after we had trained users from all of these disparate units did we realize that many had grown their own local systems for collecting their own data, mostly Excel spreadsheets. The users all balked at the idea of reentering all of their information with the Web forms, asking, “Why can’t we just upload our spreadsheets?”

Sure, we could have saved all of the spreadsheets as delimited text files and written an import procedure with CF or tried to use ODBC drivers for Excel to get at the data. But WDDX and its COM support seemed an attractive solution.

The Idea

Write a Visual Basic macro for Excel that would create a WDDX packet for the entire spreadsheet and post the packet to a ColdFusion template. The CF template would then deserialize the packet and insert each record into the database.

**“WDDX is a free,
open, XML-based
technology that
allows Web
applications
created with any
platform to easily
exchange data
with one another
over the Web”
– Wddx.org**

The biggest drawback with such an implementation was that all the contributors needed some custom software installed on their PC (with the required COM objects). After that, though, contributing all of their hard-gathered data was a breeze. If they didn't like our Web form, they could even continue to use their spreadsheets and periodically upload their records.

The Method In Excel

A VBA macro will use the WDDX COM object to serialize the spreadsheet's contents into a WDDX packet. It then uses the freeware version of Softwing's ASPTear object to send the packet to the Web server. ASPTear is used in a number of examples in the WDDX SDK, including the “Insert List of Books” Word Macro. Of course, in the Word Macro example, the point was to get information from the Web server and put it into the Word document. Our project is the opposite: to take information out of Excel and put it into the Web application.

To begin, register the two objects on the machine on which the spreadsheet will be used by placing the WDDX_COM implementation files (wddx_com.dll, xmlparse.dll and xmltok.dll wddx_com.dll) and ASPTear.dll in the Windows system directory. Then open up a MS DOS prompt, use CD to navigate to the Windows system folder where you placed the files, and type the following:

```
regsvr32 wddx_com.dll
```

and

```
regsvr32 ASPTear.dll
```

You're now ready to start Excel and create a new spreadsheet. The macro will use the column-heading values as column names in the WDDX packet, so it's important to inform users what to enter in these fields. The order isn't important, just the values. Let's say that you're inputting information about books in a fictional bookstore (original, huh?). The sheet will begin with the column names as shown in Figure 1.

From the Tools menu choose Macro, then Visual Basic Editor. After a moment, Visual Basic for Applications will appear and you can begin writing the module. Right-click on Sheet 1 in the VBA Project tree and select Insert, then Module. This will open a blank editor window as shown in Figure 2.

Before creating the Macro, VBA must know that this procedure will be referencing the COM objects that were installed earlier. To do this, select References from the Tools menu. This will open a dialog listing all available references on the machine. In the list, find and check the box next to wddx_com 1.0 Type Library. Click OK to return to the editor.

The Macro is created as a public subprocedure in VBA, so begin by typing the following in the editor pane:

```
Public Sub SendToWeb()
```

After typing and pressing the enter key, the editor will write the closing End Sub statement. Your variable declarations consist of the WDDX objects, the ASPTear object and information about the spreadsheet such as the range of used rows and columns:

```
Dim MySer As WDDXSerializer      'Allaire's WDDX serializer
Dim MyRS As WDDXRecordset        'Allaire's WDDX recordset

Dim xobj As Object, strRetVal As String 'Softwing's ASPTear

Dim HowManyRows, HowManyColumns
Dim rng1 As Excel.Range
```

Once declared, instances of each object can be created:

```
' Create instance of WDDX.Serializer, Recordset and ASPTear object
Set MyRS = CreateObject("WDDX.Recordset.1")
Set MySer = New WDDXSerializer
Set xobj = CreateObject("SOFTWING.ASPTear")
```

The recordset has been created but is empty at this point. Populating it will be done in three steps:

1. Add a column for each used column in the spreadsheet.
2. Add an empty row for each used row in the sheet.
3. Populate the values of each column row by row.

All three steps are concerned only with the area of the sheet that has been used – a range that Excel exposes with the

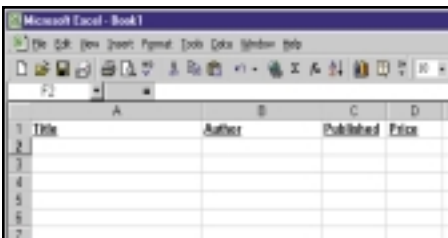


FIGURE 1: Empty sheet with column names

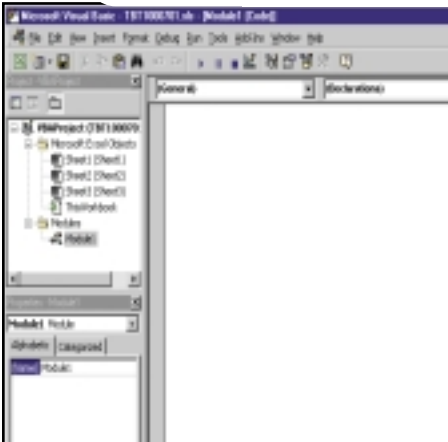


FIGURE 2: The VBA editor

UsedRange variable. With that value, count the used rows and columns through which you will later loop. Notice that the number of rows is decreased by one since your column headers are in row number one.

```
' Determine the used range of the sheet and
count the Rows and Columns
Set rng1 = ActiveSheet.UsedRange
HowManyRows = rng1.Rows.Count - 1
HowManyColumns = rng1.Columns.Count
```

The WDDX recordset needs one column for each column in the sheet. Loop through each cell in the first row and call the WDDX addColumn function to add each column name to your recordset.

```
For counter = 1 To HowManyColumns
    ThisColumn = ActiveSheet.Cells(1,
counter).Value
    MyRS.addColumn (ThisColumn)
Next counter
```

The number of rows in the recordset is the number of rows in the used range of your sheet, again decremented by one

because the first row contains our field names.

```
' Add a row for each row in the sheet
MyRS.addRows (HowManyRows)
```

So now you have the WDDX recordset with columns named for each spreadsheet column and an empty row for each spreadsheet row waiting to be populated with data. Looping through each row and within the row looping through each cell, every value is put into the recordset (see Listing 1).

Once populated, the recordset is serialized into a finished WDDX packet.

```
' Serialize it into a WDDX Packet
MyPacket = MySer.serialize(MyRS)
```

The packet is to be sent to ColdFusion via HTTP Post with the ASPTear object. If you were posting from a Web form, a name/value pair would be sent. With ASPTear, you specify the name/value pair(s) as a postData parameter in the function.

```
Const Request_POST = 1
strPostData = "WDDXPacket=" & MyPacket
strRetVal = objj.Retrieve("http://www.your-
server.com/GetWDDXFromExcel.cfm", 1, str-
PostData, "", "")
```

strRetVal will be set to whatever our ColdFusion template returns after the HTTP post is processed. This value may be used for a verification message, perhaps, informing the user that x number of rows were received and processed (hopefully) successfully. Finally, end the Sub routine by displaying the return in a message box.

```
MsgBox (strRetVal)
```

In ColdFusion

Meanwhile, back on the Web server, a ColdFusion template is patiently waiting to receive the HTTP post containing the WDDX recordset.

When the post arrives, CF first uses the CFWDDX tag to deserialize the packet out of WDDX and into a format that it can use. This is the "wddx2cfml" action. Since your

packet variable (strPostData) was named WDDXPacket in the last piece of the VBA Macro, that's the variable used in your CFWDDX tag.

```
<CFWDDX ACTION='wddx2cfml' input=#WDDXPack-
et# output='FromExcel'>
```

ColdFusion now has a recordset called FromExcel.

Back in Excel, data was added to the recordset by looping through each row and through each column in each row. Getting data out works in the same way, except that it uses the FromExcel recordset instead of the used range value.

In your real application the innermost nest would contain the database insert. For this demonstration, however, simply output the name/value pairs and display them back in Excel in our message box.

```
<cfl oop query="FromExcel">
<cfl oop index="ThisColumn" list="#FromEx-
cel.ColumnList#">
<cftoutp>#ThisColumn# = #Evaluate(ThisCol-
umn)#</cftoutp>
</cfl oop>
</cfl oop>
```

Notice that we use ThisColumn twice. By itself, #ThisColumn# outputs the field name. For its value, use #Evaluate(ThisColumn)# (meaning "the value of the column called #ThisColumn#").

And that's it! No matter how many records are in the spreadsheet or how the columns are named or ordered, ColdFusion can receive the information and do with it whatever you can dream up. Later, you might extend the process by having CF return a WDDX recordset back to Excel. This direction – from the Web to office applications – is well documented. Essentially, complex "conversations" between desktop and Web server can take place since you now have the "universal" data language of WDDX.



About the Author

Tim Buntel is a senior Web developer in Providence, Rhode Island.

tim@buntel.com

Listing 1: Adding each cell value to the WDDX recordset

```
For rngTemp In rng1.Cells
    If rngTemp.Row <> 1 Then
        If rngTemp.Column <> 1 Then
            WhatRow = WhatRow + 1
        End If
        ThisColumn = ActiveSheet.Cells(1, rngTemp.Column).Value
```

```
MyRS.setField WhatRow, ThisColumn, rngTemp.Value
End If
Next rngTemp
```

CODE LISTING



The code listing for this article can also be located at www.ColdFusionJournal.com

inteliant

www.inteliant.com

macromedia

www.macromedia.com

macromedia

www.macromedia.com

Hors d'Oeuvres Anyone?

An assortment of ideas to whet your appetite



BY
HAL
HELMS

This month I've taken the liberty of assembling a small collection of unrelated ideas that I think you might enjoy as ColdFusion appetizers.

Mind Maps

Lately I've been playing with a new piece of software called Visual Mind (www.visual-mind.com) that lets the user...ummm...well, it's hard to define. Visual Mind belongs to a category of products called *mind-mapping software*. Perhaps the best way to explain it is to imagine being able to create free-form trees of information. Each node in the tree can be a text note, a Web page, a document on your hard drive or an image. Take a look at Figure 1, for example.

You can export this figure into a text file that translates the visual hierarchy into an outline, such as the following:

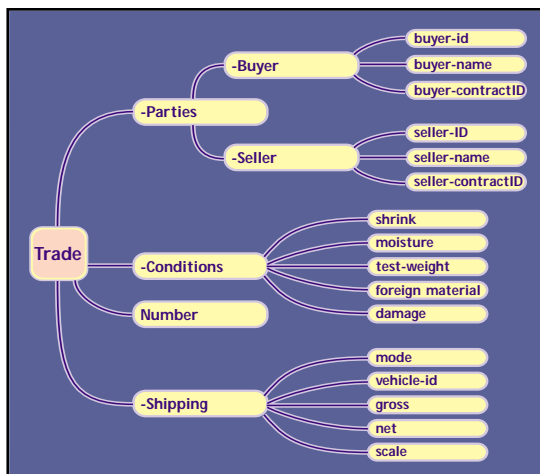


FIGURE 1: Graphical representation of XML.DTD

```
Trade
1. Parties
1.1 Buyer
1.1.1 buyer-id
1.1.2 buyer-name
1.1.3 buyer-contractID
1.2 Seller
1.2.1 seller-id
1.2.2 seller-name
1.2.3 seller-contractID
....
```

I've found it useful for creating XML Document Type Definitions – those templates for XML documents – since it provides an easy way to map the structures for a given DTD. For domain experts this graphical representation is much easier to make sense of than the DTD itself. Once I've got it right, creating the DTD is simple stuff. I imagine this tool could have a wide range of uses.

Ini Files

I've gotten so used to using databases for persistent storage that I almost never consider any alternatives. For some applications, or in certain environments, databases may not be the best choice.

One option to consider is the ini file used by so many Windows programs to store information. Ini files are organized like this:

```
[section name]
someName=someValue
anotherName=anotherValue
```

You can use your own ini file easily enough by using two of the more obscure ColdFusion functions: `SetProfileString()` and `GetProfileString()`.

To set name/value pairs with `SetProfileString()`:

```
<cfset
SetProfileString('c:\inetpub\www-
root\test\preferences.ini', 'Compa-
ny', 'name', 'ABC, Inc.')>
```

This will create an entry that looks like this:

```
[Company]
name=ABC, Inc.
```

If the file you specified doesn't already exist, ColdFusion will create it for you.

To read specific info use the `GetProfileString()` function:

```
<cfoutput>
This copy of Make-A-Zillion software
licensed to
#GetProfileString('c:\inetpub\www-
root\test\preferences2.ini', 'Compa-
ny', 'name')#
</cfoutput>
```

SKU	QUANTITY	DESCRIPTION	IMAGE	PRICE
18874	1	How to Make a Zillion Dollars in Stocks	ZD5886.gif	\$49.95
98833	1	Filing for Bankruptcy for Dummies	BR9948.gif	\$29.95

TABLE 1: Items in a typical shopping cart

digital nation

www.dedicatedserver.com

HAL.HELMS@TEAMALLAIRE.COM

developersnetwork

www.developersnetwork.com

Lock It or Lose It



BY
BEN
FORTA

ColdFusion locking, and the correct use of the <CFLOCK> tag, seems to baffle even the most seasoned CFers among us

We all know that locking is important. Most of us even understand why locks are needed. But exactly where to use a lock, which lock type to use and what code to put within the lock remains confusing at best.

Part of the confusion stems from changes Allaire made in ColdFusion 4.5 that in turn changed the recommendations and suggested practices. Indeed, even my own recommendations changed with that release (as many of you CFUG members are quick to point out). And so, at the request of several of you, and because I've helped contribute to the confusion, I'll cover these topics in this column and try to set the record straight.

Variables

Locks are used primarily with variables, so let's start there. Variables are kind of virtual containers in memory, containers that are used to store data. Look at the following code:

```
<CFSET first_name="Ben">
```

The <CFSET> tag creates (or overwrites) a variable, in this case a variable named first_name. first_name can be thought of as a container located somewhere in the memory of the computer, a container that now has the name "Ben" in it. To access the data in the container you simply refer to the container by name, like this:

```
<CFOUTPUT>#first_name#</CFOUTPUT>
```

In this example I used a simple variable. I could just have easily placed an array or list in that container, or even data as complicated as an array of structures containing arrays of structures, and so on.

Regardless of the type of data, one thing is consistent: you refer to the container (the variable) by a unique name, and that name provides access to the contents of the container at the moment it's requested.

Understanding Threads

Before I go further, one other topic must be mentioned briefly – threads. ColdFusion is a high-performance application server; it's designed to process many requests at once. It does this by running lots of concurrent tasks within the application server, and each one handles a single request at any given time. These tasks are known as "threads," and ColdFusion is a multithreaded application – in other words, ColdFusion is designed to perform multiple tasks concurrently. (There's actually much more to threads than that, but this explanation is adequate for the issue at hand.)

Simultaneous Variable Access

ColdFusion supports several different data scopes. "Scope" defines

the life span (persistence) and visibility of data. Let's take a quick look at five scopes:

1. **VARIABLES:** Used for data that needs no special persistence. Data in the VARIABLES scope persists for the duration of the processing of a request and is automatically destroyed once the request has completed. The data is visible only within the thread processing that request.
2. **SESSION:** Used for session variables, for data that relates to many requests that together make up a session. Data in the SESSION scope persists until the session times out. The data is visible to any thread that processes requests for that session, and it's entirely possible that multiple threads will process requests for the same SESSION (even though a SESSION is mapped to a single client).
3. **CLIENT:** Also used for session variables, but CLIENT is a little different from SESSION. Unlike SESSION variables, CLIENT variables aren't stored in memory. Rather, they're stored in a database (a database of your choice or the Windows registry, which is also a database of sorts). Data in the CLIENT scope persists until the client session times out. The data is visible to any thread that processes requests for that client session, and it's entirely possible that multiple threads will process requests for a single CLIENT session (even though a CLIENT session is mapped to a single client).
4. **APPLICATION:** Used for variables that are shared across complete applications. Data in the APPLICATION scope is visible to all threads processing requests for that application.

“
Locking is an important CF feature, and one that serious developers must use in their applications

corda technologies

www.corda.com

5. **SERVER:** Used for variables that are shared across all applications running on the ColdFusion server. Data in the SERVER scope is visible to every thread on the server.

The variable `first_name`, created earlier, was created in the VARIABLES scope. As explained above, this scope is processed by a single thread, and as soon as that thread has completed processing the request, the variable is destroyed. As such, there is absolutely no way more than one request could access the data in the `VARIABLES.first_name` container at the same time.

But other scopes behave differently. The following code creates a variable in the SESSION scope:

```
<CFSET SESSION.first_name="Ben">
```

As explained above, SESSION variables can indeed be processed by multiple threads at once. If you use frames, if the user hits the refresh button, if the underlying network makes retries – there are lots of conditions that could cause the same SESSION to be processed by more than one thread at any given time.

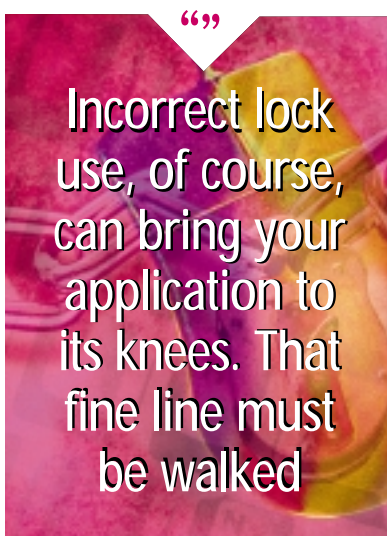
This is where things get dangerous. Let's go back to our container analogy. If you were to put data into a container at the exact moment someone else was doing so, what would happen? Both writes couldn't occur at the same time, so something would get lost – or worse, the container itself could become corrupted. If the `<CFSET>` statement above was executed at the exact same time as another `<CFSET>` statement that was updating the same variable, you'd likely corrupt server memory. If you're lucky, you'll just throw an error, but you could also negatively impact server operation as a whole.

And it's not just SESSION variables that are affected. APPLICATION and SERVER scope variables are even more susceptible to this corruption as they're always shared. (CLIENT variables, however, aren't susceptible as they're stored in a database; the database handles concurrency issues itself.)

Using Locks

How do you get around this problem? The answer is to use a lock. A lock does just that – it locks a block of code (a block containing a `<CFSET>` statement, for example). Going back to our container analogy, a lock acts as a guard monitoring access to the container's contents. The guard's job is to line up all access requests in the order they're received, granting admission one request at a time, and only after the previous access request has completed.

In other words, locks can arbitrate code execution across multi-



ple threads, pausing execution as needed. And yes, this could slow down your application, but considering the alternative it's a small price to pay. Accessing a variable (writing or reading) while it's being written by another thread is asking for trouble.

The next code snippet sets the same SESSION variable once again, but this time locking it for the duration of the update:

```
<CFLOCK SCOPE="SESSION" TYPE="EXCLUSIVE" TIMEOUT="10">
  <CFSET SESSION.first_name="Ben">
</CFLOCK>
```

Locking is implemented using the `<CFLOCK>` tag, and any code between the `<CFLOCK>` and `</CFLOCK>` tags will be locked. The SCOPE attribute specifies the scope to be locked by specifying SESSION as the scope. We're instructing ColdFusion to lock only the code execu-

tion for a particular SESSION. We wouldn't want to lock all sessions as that would cause other operations to pause unnecessarily (they wouldn't be updating this SESSION anyway). The TYPE attribute specifies the lock type. EXCLUSIVE means that no other operations on the specified SCOPE will be allowed while the lock is being processed. The TIMEOUT specifies the maximum time that ColdFusion should wait when trying to acquire a lock. If that timeout is reached before the lock can be acquired (perhaps because other threads have the same scope locked), the entire `<CFLOCK>` code block is skipped (and an exception is thrown).

To lock the APPLICATION scope you'd simply specify `SCOPE="APPLICATION"`. Doing so would lock the APPLICATION scope so any other attempt to access APPLICATION data would be paused. The same is true for SERVER.

It's important to note that `<CFLOCK>` will do its job if all appropriate code is enclosed within `<CFLOCK>` tags. If somewhere in the code you had a `<CFSET>` statement that didn't use a `<CFLOCK>`, it could access the variable even though it was locked. For locking to work, all accesses must be managed by `<CFLOCK>` statements.

SCOPE vs NAME

In the example above I used `SCOPE="SESSION"` to lock my `<CFSET>` statement. Three scopes are supported: SESSION, APPLICATION and SERVER. Specifying a SCOPE of SESSION locks any other accesses for the same SESSION only. Specifying a SCOPE of APPLICATION locks any other accesses for the same APPLICATION (as named in the `<CFAPPLICATION>` tag, usually in `APPLICATION.CFM`). Specifying a SCOPE of SERVER locks any other accesses for SERVER scope locks server-wide.

ColdFusion also supports locking by NAME. Using this method, you provide a name to identify the activity performed in the locked code, and only locks with the same NAME will be locked. Exactly what operations are locked within the lock is entirely up to you. All `<CFLOCK>` does is ensure that no two blocks of code with the same NAME are exe-

cuted at once. Using NAME gives you a greater level of control over lock granularity, but with that control comes additional risk. If you mistakenly use different names for two locks that access the same data, you won't be locking at all.

The ability to lock code by scope was introduced in ColdFusion 4.5, and it's the preferred way to lock code that accesses potentially shared variables.

Read-Only Locks

Locking is really an issue only when variables are being written to. Going back to our container analogy, if multiple users looked into the container at the same time to see what was in it, no harm would be done. The same is true of read access to shared variables.

Some languages support the use of constants, special variables that are actually not variable at all as they can't be changed. ColdFusion has no concept of constants, so CF developers typically create variables in the APPLICATION scope (usually in the APPLICATION.CFM file surrounded by a <CFIF NOT IsDefined(> check) and are careful never to overwrite them. If an application contained variables like this, variables that were never updated (after initial creation), you wouldn't really need to lock them at all. But you'd have to be 100% sure that an update wouldn't occur, realizing that there's nothing you can do programmatically to prevent that.

What to do? Locking all read accesses (every time you refer to #SESSION.first_name#, for example) with exclusive locks imposes a significant performance loss, and the risk may not be worth it. So you could opt not to lock variable reads.

But there's always the chance that someone will edit the code, and the variable that was never supposed to be updated...well, what if some new code now updated it?

To address this problem, ColdFusion supports an additional lock type, READONLY. A READONLY lock doesn't actually lock anything unless an EXCLUSIVE lock is being processed at the same time. Only then will the READONLY lock pause until the EXCLUSIVE lock has completed. In other words, READONLY

locks have no real performance hit associated with them. They are essentially ignored until an EXCLUSIVE lock is in effect.

Other Operations Needing Locks

Variables aren't the only things that need locking. Any code with potential concurrency issues should be locked. Examples of this include:

- Accessing files with <CFFILE> if other processes could be accessing the same data file
- Calling code that isn't multi-thread safe (some CFX tags, for example)
- Connecting to remote sites via <HTTP> if those sites don't allow concurrent connections

In all of these examples, locking the code block can avoid concurrency problems. But instead of locking by SCOPE, these operations should be locked by NAME.

Locking Tips

Locking is important and must be used. But locking slows your application, as already mentioned. Locks must be used carefully, and they must never be overused. Here are some pointers to keep in mind:

- Don't lock code unnecessarily, but don't create and drop locks too frequently. It's a fine line to walk, but if you find yourself needing to lock two variables with some other lengthy processing in between them (that doesn't need locking), you might be better off using two locks so you don't keep locks active when they're not needed.
- If you find yourself having to perform complex operations on locked variables (for example, complex string processing, looping or WDDX decoding), consider making local (VARIABLES) copies of the data and performing the processing on the local copy, then using a lock only when saving the local copy back to the shared variable.
- APPLICATION locks should be used sparingly as they typically apply to lots of code. If you need to lock only part of the APPLICATION scope, consider using the NAME attribute instead of SCOPE. This will give you more

granular control over exactly what gets locked and when, which in turn can prevent unnecessary locking (or unnecessarily long locking times). Of course, as explained earlier, using NAME comes with a risk. You must be sure that different names aren't used for code that accesses the same data. (The same thing applies to SERVER variables.)

- ColdFusion 4.5 supports automatic locking modes in which ColdFusion locks variables for you. There's a bit of overhead in using auto-locking, plus you'll lose the potential performance gains that could be attained by more granular locking. As a rule, if performance is an issue (and when isn't it?), don't use these options. You'll be able to squeeze a bit more performance out of your application by doing it yourself.
- ColdFusion 4.5 also supports a mode called *Full checking*. In this mode no locking occurs automatically. Instead, ColdFusion throws an error if a lock isn't used, helping you eliminate potentially missed locks. But this option can't be used with NAME locks – it'll always throw errors on those.

The common theme is that locks should be used, but they must be used carefully. And careful use requires a good understanding of what locks are, what they do and how your application should use them.

• • •

Locking is an important ColdFusion feature, and one that serious developers must use in their applications. Without locking there's a very real risk that data corruption will occur, and this can impact server stability.

Incorrect lock use, of course, can bring your application to its knees. That fine line must be walked. Yes, there are performance penalties involved, but every decision involves some kind of trade-off.

What you do is your choice. My advice? Lock it or lose it.



ABOUT THE AUTHOR

Ben Forta is Allaire Corporation's product evangelist for the ColdFusion product line. He is the author of the best-selling ColdFusion 4.0 Web Application Construction Kit and its sequel, Advanced ColdFusion 4.0 Development, as well as Sams Teach Yourself SQL in 10 Minutes. He recently released Allaire Spectra E-Business Construction Kit, and is now working on books on WML and JSP.

BEN@FORTA.COM

Live Monitoring of User Sessions

Avoid the need for late-night pizzas...
by knowing how many users are
currently active in your apps

BY
CHRISTIAN
SCHNEIDER



During a recent intranet project, I encountered the need for live reports of all currently open user sessions within an application.

The goal was for me to be able to see how many users were currently online with an open session when rolling out new templates from the development server to the production server. That way I would be able to determine whether it was a good idea to update the live production server or if it would be better to wait until the late evening (of course, I'd probably need to call the local pizza service, instead of having a nice dinner with my girlfriend). It would put an end to the numerous nights behind the PC if I could develop some kind of a live-session monitor that enabled me to see how many users were currently online with running sessions.

Overview of Sessions

Sessions are frequently used by ColdFusion developers to track a certain state of a connection over multiple pages. Since HTTP is by nature a stateless protocol, meaning it has no built-in mechanisms to track a session over multiple pages (instead, each page request opens a new connection to the host), the guys from Allaire opted to use cookies and issue unique IDs to each client and incorporate this into the popular session framework. That way developers can simply set any variable in one template into the session scope: `<cfset SESSION.userName = "John">` and access them in another template called by the client afterwards. This session framework, as part of the CF Web application framework, is activated within the `Application.cfm` file by using the following statement:

```
<cfapplication name="My_Test_App"
clientmanagement="yes" sessionmanagement="yes" sessiontimeout=#CreateTimespan(0, 0, 20, 0)#>
```

This statement sets the default timeout of a session to 20 minutes by using the `CreateTimespan()` function. It means that after 20 minutes of inactivity the client's session is closed. When returning to the application, this timeout-event should be caught and the user sent back to the login page. Setting a session timeout is mainly used for security reasons; it's also used on large-scale sites since keeping sessions open for a long time would produce too much overhead on the server. Now I'll present a solution on how to live-monitor these sessions.

A Simple but Effective Solution

As difficult as it sounds to develop a live monitor of all open sessions, it was (to my own surprise) done quite easily with the use of ColdFusion mechanisms and techniques. All we need are three ingredients: applications-scoped variables, structures and the `Application.cfm` file.

Application-Scoped Variables

The variables set in the application scope, like `<cfset APPLICATION.systemColor = "Blue">`, are accessible (read and write) by all clients accessing the application, as opposed to *session-scoped* variables that are accessible (read and write) only by each client separately.

Structures

These kinds of variables are like associative Arrays, which have a set of keys in them mapped to certain values. They could be described as a bunch of key-value pairs held within one variable that are set the following way:

```
<cfset myStruct = StructNew()> <cfset
myStruct["color"] = "Blue">
```

Alternatively the last statement could be coded another way, using the ColdFusion structure functions:

```
<cfset dummy = StructInsert(myStruct,
"color", "Blue", true)>
```

where the last boolean parameter (here true) indicates whether or not existing keys should be overwritten. Note that CF has a very rich set of structure functions, including a mechanism of looping over structures that we'll also be using.

Application.cfm File

Our next basic ingredient is the `Application.cfm` file, the root for enabling the ColdFusion Web application framework. It has a useful characteristic: its content is executed before any page request within an application.

Having listed these three ingredients, I can now tell you the recipe for a live-session monitor. In brief: we need to set some kind of flag when a user is active. When we timestamp this flag we can also determine if the user's session has timed out (and is therefore no longer active). CF treats every page request within an application as an event to set the session timeout ticker back, which means it sees that the user is still active and his or her session should be kept active too. The easiest way to set the flag is to do so in the `Application.cfm` file, because this file is executed on every page request.

As we wish to track every user of an application, this flag should be placed into the application scope so

conceptware

www.conceptware.com

The screenshot shows a web browser window with the title 'Live Session-Tracker Report'. The address bar shows 'http://localhost:8080/cfreport'. The report displays a table with two columns: 'User (by IP-address)' and 'Session-Status'. The table lists 20 users, each with an IP address and a session status indicating inactivity time. The background of the rows is color-coded based on inactivity time: blue for 0-1 min, yellow for 2-5 min, and orange for 6-10 min. At the bottom, it says '20 Users online'.

User (by IP-address)	Session-Status
10.120.18.157	inactive since 0 mins
10.120.18.52	inactive since 0 mins
164.210.88.102	inactive since 0 mins
10.82.190.72	inactive since 4 mins
164.210.128.75	inactive since 6 mins
10.88.110.52	inactive since 6 mins
10.88.110.192	inactive since 6 mins
10.88.110.12	inactive since 6 mins
10.82.190.12	inactive since 2 mins
164.210.128.15	inactive since 6 mins
10.120.180.157	inactive since 6 mins
127.0.0.1	inactive since 0 mins
10.120.120.152	inactive since 1 mins
160.10.180.187	inactive since 1 mins
10.80.10.52	inactive since 1 mins
160.10.180.122	inactive since 1 mins
10.88.10.52	inactive since 5 mins
164.210.88.15	inactive since 7 mins
10.120.18.50	inactive since 6 mins
164.210.128.93	inactive since 6 mins

20 Users online

FIGURE 1: Sample view of the session monitor

it can hold information about all current active users. Besides (as you might already guess from my ingredients list), this flag is a structure logging every active user's timestamp with some unique part as the key and the current timestamp as its value. This unique key could be a user ID derived from the user's login or just the user's IP address (see Listing 1).

Of course, if you run a site that asks every user to log in (as most intranet applications do), taking the user's login ID here would be nice, because that way your session monitor would show you not only the IP addresses of all active users but also their actual user names. I decided to take the user's IP address as the key in Listing 1 for demonstration purposes and to keep the code snippet general so you can take it and run it on your site.

The rest is a simple reporting page that loops over the mentioned structure and checks whether each logged session is still active or has timed out, and presents the neatly formatted results. As we have the timestamp along with the user's IP address, it's also possible to set some thresholds for color-coding the report according to how long a session was inactive.

Explaining the Code

Now I've presented my idea for developing the session monitor, let's

look inside the code. Basically all we need is a code snippet to place into the Application.cfm file shown in Listing 1 and a simple reporting page that presents a nicely formatted session report (see Listing 2).

First I'd like to comment the Application.cfm snippet shown in Listing 1. The first cfset statement sets a locally scoped variable, theTimeout, to a time span by using ColdFusion's CreateTimeSpan() function. This is necessary to define the timeout for CF's session framework activated in the next statement. You might wonder why I decided to use a locally scoped variable instead of directly using CreateTimeSpan() in the cfapplication tag, but as I need the timeout value in the reporting page to determine when a user's session has timed out, setting a variable and referencing it makes more sense.

The next block of code is the session monitor's heart; it's surrounded by a cflock statement because proper locking of application-scoped variables is a good style of writing code and a must to guarantee consistency between simultaneous accesses in large-scale sites. Inside the heart the session monitor generates the structure holding the tracked information (APPLICATION.SessionTracker) if it doesn't already exist. The last and most important statement:

```
<CFSET dummy = StructInsert(APPLICATION.SessionTracker, CGI.REMOTE_ADDR, Now(), true)>
```

takes the user's IP address as a key and the current time as its value and inserts this into the session monitor's structure. If the key (meaning the user) already exists in the structure, it's automatically updated. This feature is made available by the boolean parameter true of StructInsert()'s parameters. As this snippet is executed on every page request, it continuously updates the structure that holds all active users and is fed into the reporting page (shown in Listing 2 and explained next).

The code in Listing 2 is just a sample reporting page that uses the session monitor's application-scoped structure holding all active users with their last activity time-stamped. It shows an HTML table listing all active users and their inactivity time. This report is color-coded according to how long a user was inactive (see Figure 1). The threshold values for coloring the report might be individualized depending on how granular you want the report to be colored (just take mine as sample values). Besides the surrounding cflock statement to guarantee consistency for simultaneous accesses and the HTML code to format the table, Listing 2 contains the following interesting code: inside the table the reporting page loops over the session monitor's structure and determines the time of each user's last activity to decide whether the user's session is still active or has timed out. This is achieved by comparing the time of the user's last activity plus the timeout value against the current time. If the user's session has timed out, the entry is removed from the session-tracker using the StructDelete() function. Otherwise the still-active session is written onto the report along with the user's inactivity time, colored according to how long he or she was inactive. That way you get a live report showing all active users (see Figure 1).

Introducing OnRequestEnd.cfm

Introduced with ColdFusion 4.01, OnRequestEnd.cfm is the counterpart of Application.cfm. As the name

suggests, OnRequestEnd.cfm is executed at the end of each request, like Application.cfm at the beginning. This means you're free to decide whether you'd like to place the session monitor's heart (the block of code inside the cflocks in Listing 1) into the Application.cfm or OnRequestEnd.cfm file.

Further Ideas

Having presented my idea of a simple session monitor, I'd like to conclude with a few thoughts and enhancements based on the ability to track user sessions.


Of course, the first (and probably easiest) enhancement would be to define a more detailed report that could be sorted (by inactivity time) and filtered (by usergroups). This should be implemented easily using the handy CF functions for structure, array and list handling. I decided not to implement it here because I wanted to keep the code simple.

If your monitored application supports a defined group of users that have to log in, another straightforward enhancement would be to log by user IDs instead of IP addresses. This enables grouping based on how many users from any given department are currently online.

Using the Application.cfm file more intensively with the awareness of who is currently active, it's also possible to develop some kind of instant messaging between online users by setting an application-scoped message structure indexed by user IDs with the messages as their values. Whenever a user accesses a page of the application that has instant messaging, this queue is checked for messages for the current user.

Knowing how many users are currently active in your apps is also valuable information for licensing and security issues. For example, you can prevent applications you've

sold from being used by more simultaneous online users than your customer has purchased licenses for (depending on your licensing model). Knowing how many simultaneous users are accessing your application can also be used to initiate certain mission-critical tasks whenever too many users are active, such as automatically caching queries, activating the sleeping backup server to become part of the productive cluster or just sending an e-mail warning to your system administrator to watch the servers.

These are only a few aspects of this topic. I'd be interested in any feedback and ideas you may have so I can incorporate them and make the session monitor a handy feature every application should have. 

ABOUT THE AUTHOR

Christian Schneider is a self-employed ColdFusion developer with over three years of intensive experience in developing CF-based intranet applications for banks and logistic corporations.

MAIL@CHRISTIAN-SCHNEIDER.DE

Listing 1: Application.cfm

```
<cfset theTimeout = CreateTimespan(0, 0, 30, 0)>

<!--- Set Application name and options --->
<CFAPPLICATION NAME="My_Test_App"
CLIENTMANAGEMENT="YES"
SESSONMANAGEMENT="YES"
SESSONTIMEOUT=#theTimeout#>

<!--- Session-Tracker Code --->

<cflock name="#APPLICATION.applicationName#"
type="Exclusive"
timeout="20"
throwontimeout="Yes">

<!--- If Session-Tracker does not exist,
generate it --->
<cfparam name="APPLICATION.SessionTracker"
default=t=StructNew()#>

<!--- Log current user --->
<CFSET dummy = StructInsert(APPLICATION.SessionTracker,
CGI.REMOTE_ADDR, Now(), true)>

</cflock>
```

Listing 2: SessionReport.cfm

```
<html>
<head>
<title>Live Session-Tracker Report</title>
</head>
<body><basefont face="Arial">
<cfoutput>

<h3>Live Session-Tracker Report</h3>

<cflock name="#APPLICATION.applicationName#"
type="Exclusive"
timeout="20"
throwontimeout="Yes">

<table border="2" cellpadding="0" width="95%">
<tr bgcolor="#cccccc">
<td><b>User (by IP-address)</b></td>
<td><b>Session-Status</b></td>
</tr>

<CFLOOP collection=#APPLICATION.SessionTracker#
```

```
item="aUser">
<cfset onlineSince = StructFind(APPLICATION.SessionTracker, aUser)>

<CFIF DateCompare(onlineSince+theTimeout, Now()) EQ 1>
<!--- User's last activity lies within session-timeout,
so his/her session is active --->

<cfset inactiveSince = DateDiff("n", onlineSince,
Now())>

<!--- The threshold for coloring the report
may be set individually: --->
<cfif inactiveSince LTE 2>
<cfset theColor = "Red">
<cfelseif inactiveSince LTE 5>
<cfset theColor = "Yellow">
<cfelse>
<cfset theColor = "Cyan">
</cfif>

<!--- Output of current user in report --->
<tr>
<td bgcolor="#eeeeee">#aUser#</td>
<td bgcolor="#theColor#">
inactive since <b>#inactiveSince#</b> mins
</td>
</tr>

<CFELSE>
<!--- User's session has timed-out, so we can
delete his IP from the Session-Tracker --->

<cfset dummy = StructDelete(APPLICATION.SessionTracker, aUser)>

</CFIF>
</CFLLOOP>

</table><p>
#StructCount(APPLICATION.SessionTracker)# Users online

</cflock>

</cfoutput>
</body>
</html>
```

CODE LISTING



The code listing for this article can also be located at www.ColdFusionJournal.com

Dynamic Goes Static

Create successful dynamic applications that can be turned into static pages with ColdFusion

BY
DUSTIN
SMITH



For many developers there comes a time when you think your options are limited. You can “risk” a completely dynamic application or use it as sparingly as possible, thus limiting your capabilities.

How about combining the advantages of a static page with those of a dynamic page through the power of ColdFusion? This article shows you how.

As you probably know, static pages have many advantages over dynamic pages, including:

- **Dependability:** After you’ve created a static page, the output will generally remain the same, practically eliminating the possibility of an embarrassing error.
- **Speed:** Static cuts out the time-consuming step of processing a dynamic template, which reduces the response time of the page.
- **Search engine indexing:** For the most part, search engines can index ColdFusion pages. However, when you have a veritable string attached (like `index.cfm?id=1`), the page can’t be indexed.

Where You Can’t Use Static

Static pages can be updated only when triggered or scheduled to do so; therefore change-critical or heavily updated pages should probably remain dynamic. Pages that involve user-supplied information (searching or client input-dependent pages) can’t be static.

How to Use ColdFusion Functions to Generate Static Pages

By merely using the `CFHTTP` and `CFFILE` functions, you can generate static pages. You’ll need:

1. A directory to store templates that will have their exact output generated into static pages. I’ll use the directory `c:/root/temp/` for this example.
2. A file that does the actual genera-

tion of static pages but isn’t accessible to all users. For this example I’ll name the file `dyn2static.cfm`.

The sample generation of a basic static page is:

```
<CFHTTP
URL="http://www.domain.com/temp/static_page_1.cfm" METHOD="Get">
<CFSET IndexBuild = "#CFHTTP.FileContent#"> </CFHTTP>

<CFFILE ACTION="WRITE"
FILE="c:/root/index.cfm"
OUTPUT="#IndexBuild#"
NAMECONFLICT="overwrite">
```

Easy enough! You now have a static page in your main directory (`c:/root/`) that mirrors the dynamic page in your temporary directory (`c:/root/temp/`). *Note:* Make sure your links are mapped to the directory in which you’ll be building the new files; otherwise they won’t work in a new directory. The generated file (for this example I used `index.cfm`) doesn’t have to have the “.cfm” extension but can use “.html”; however, I like to use it just to show that I’m using ColdFusion.

More Complicated Areas

The preceding example shows how you could morph a cumbersome dynamic page into a static page. But how do you solve the problem of veritable strings on your dynamic files? Here’s how: when you create a dynamic page with a dynamic link, you have a receptor page that takes the veritable string and processes it. For example, on `static_page_2.cfm` you

may have something that looks like this:

```
<CFQUERY DATASOURCE="LINKS"
NAME="GetLinks"> SELECT Link_Name,
Link_ID
FROM
Links ORDER BY Link_Name </CFQUERY>

<UL> <CFOUTPUT QUERY="GetLinks">
<LI><A
HREF="static_page_3.cfm?link_id=link_id#">link_name#</a> </CFOUTPUT>
</UL>
```

When you use generated links you can’t index them with search engines, which limits your exposure. Logically, there’s only one other solution to this problem: use separate pages for each link. Of course, to manually create a separate page for each link would be more of a problem than it’s worth. Here’s how to combine the existing method of dynamic links with the generation of static pages.

First, edit the dynamic template to make links to static files. Change the above code to something like this:

```
<CFQUERY DATASOURCE="STORE"
NAME="GetProducts"> SELECT
Product_Name,
Product_ID FROM Products ORDER BY
Product_Name </CFQUERY>

<UL> <CFOUTPUT QUERY="GetProducts">
<LI><A HREF="#Replace(product_name,
" ",
"_" )#/">#product_name#</a> </CFOUT-
PUT> </UL>
```

which would return something like this:

As you can see, instead of a number, a directory with the link's name represents the link. The only difference is that the spaces between each word are replaced by an underscore; this is because most servers (and clients) don't respond to spaces in file names. Another thing to be careful of is names over 64 characters that could trigger an error. To resolve this, use the `left()` function:

If you prefer, you could also generate pages/directories based on the product ID, which you can be sure is always unique.

For this example I chose to use the field's name as a common link. However, you could use another field if it's more suitable for your project. Just make sure it's something unique from each field. You don't have to generate a directory for each link, as I'm doing; you can choose alternative methods such as individual .html files. I chose this method since it would be the most informative for any situation.

Now that you've made the page "static ready," we'll work on the

It's usually a good idea to include some type of status output so you can pinpoint errors if they occur (see Listing 2).


After this template runs, it creates a directory for each product name and an index file inside each directory. As I mentioned before, this setup wouldn't work if there were products with the same name. It would probably be wise, although messy, to use the ID field as the name of the file.

You can now create successful dynamic applications with ColdFusion that can be turned into static pages. There are many different extensions of this ability that you can use. I'll suggest a couple of my own.

You can automatically schedule the ColdFusion server to run the template that builds the static pages by using CFSCHEDULE or the scheduling feature in the Administration panel. If you schedule the page to be processed, you won't have to load the page manually; it'll be done automatically at a given time or date. You can also run additional preprocessing scripts to ensure that your pages don't have

errors and other bad outputs. New versions of the ColdFusion server already have a function in the Administration panel to create static pages from dynamic ones; however, you can't check for errors before processing and you'll need to follow the previous examples for pages with dynamic links.

A good way to ensure that your visitors will never encounter an embarrassing error is to scan for them before you build the static page. In the code in Listing 3 the returned template's HTML is scanned for a ColdFusion error. If no error messages are found, the page is built. If an error is encountered, no static page is generated and an e-mail is sent to the administrator advising him or her to correct the problem (see Listing 3).

I've now covered most areas of static page generation with ColdFusion. If you have any problems with any of the topics covered above, e-mail me and I'll be glad to help. 

Dustin Smith is the founder of Digital Studios, an Internet consulting company. A member of the Allaire Alliance for ColdFusion programming, he has worked with ColdFusion for the past two years.

SALES@JEXNET.COM

```
<CFQUERY DATASOURCE="Store" NAME="GetProducts"> SELECT Product_Name,
Product_Id FROM Products </CFQUERY>

<CFLLOOP QUERY="GetProducts">

<CFSET ProdName = "#Replace(GetProducts.Product_Name, " ",
"_"#)">

<CFHTTP
URL="http://www.domain.com/temp/products.cfm?product_id=#Get-
Products.pro
duct
_id#" METHOD="Get"> <CFSET GetPage = "#CFHTTP.FileContent#">
</CFHTTP>
```

```
<CFOUTPUT> P i n g i n g   P r o d u c t   [ I D = # G e t P r o d u c t s . p r o d u c t _ i d # ] < B R >
< / C F O U T P U T >

< C F I F   F i l e E x i s t s ( " c : \ r o o t \ s t o r e \ # P r o d N a m e \ # i n d e x . h t m l " )   I S
" N O " >
< C F D I R E C T O R Y   A C T I O N = " C R E A T E "   D I R E C T O R Y = " c : \ r o o t \ s t o r e \ # P r o d -
N a m e # " >
< / C F I F >

< C F F I L E   A C T I O N = " W R I T E "   F I L E = "   c : \ r o o t \ s t o r e
\ # P r o d N a m e \ # i n d e x . h t m l "
O U T P U T = " # G e t P a g e # "   N A M E C O N F L I C T = " o v e r w r i t e " >
```

```
<CFHTTP URL="http://www.domain.com/page.cfm" METHOD="Get">
<CFSET
GetPage =
"#CFHTTP.FileContent#"> </CFHTTP>

<CFIF #FindNoCase("Error Occurred", GetPage)# = 0>

NO Error. Build Pages

<CFELSE>a

<CFMAIL TO="admin@domain.com" SUBJECT="urgent, error!">
Fix me!
</CFMAIL>

</CFIF>
```

DDDDDDDDDD

The code listing for
this article can also be located at
www.ColdFusionJournal.com

A Beginner's Guide to ColdFusion

Using Forms to Add or Change Data

Part 2

A primer in dynamic page development



FROM THE BOOK
BY BEN FORTA

Deleting Data with ColdFusion

Unlike adding and updating data, ColdFusion provides no efficient way to delete data. DELETE is always a dangerous operation, and ColdFusion developers didn't want to make it too easy to get rid of the wrong data.

To delete data in a ColdFusion template you must use the SQL DELETE statement, as shown in Listing 10 [Ed. Note: Listings 1–9 and Figures 1–4 can be found in Part 1 (CFDJ, Vol. 2, issue 7)]. The code first checks to ensure that an employee ID was passed; it terminates if the EmployeeID field is not present. If an employee ID is passed, a <CFQUERY> is used to pass a SQL DELETE statement to the ODBC data source.

The following code deletes the record for the employee ID passed:

```
DELETE Employees WHERE EmployeeID =  
#EmployeeID#
```

If EmployeeID were 7, the code would translate to the following and the employee with an employee ID of 7 would be deleted from the Employees table:

```
DELETE Employees WHERE EmployeeID = 7
```

Reusing Forms

You can now add to as well as update and delete from your Employees table – but nothing lasts. Just when you thought you could relax and take a day off, Human Resources needs you to provide access to additional table columns.

You start to modify both the Employee Add and the Employee Update forms. You make sure that the additional five fields are added to both templates and that they are in the same order, spelled the same way, and are the exact same length.

Then you realize that you are doing everything twice! There is really very little difference between the Add and Update forms, except that one needs existing values prefilled for updating. The form itself is identical.

With all the effort you have gone to

in the past articles to prevent any duplication of effort, this seems counterproductive.

Indeed it is.

The big difference between an Add and an Update form is whether the fields are prefilled to show current values. Using ColdFusion conditional expressions, you can create a single form that can be used for both adding and updating data.

A new and improved Add and Update form is shown in Listing 11.

Understanding Conditional Forms

Now analyze Listing 11. The first thing you do in it is determine if an insert or an update is required. How can you know that? An employee ID must be passed in order for a record to be updated; otherwise ColdFusion would have no idea which record needs updating. It makes no sense to pass an employee ID when adding a row; the new employee's ID will be assigned when the data is actually inserted into the table.

You can therefore make a safe assumption that this is an update operation if an employee ID is present; if not, it's an addition.

The first line in the template checks to see if EmployeeID was specified:

This article has been adapted from the second part of Chapter 13 of *ColdFusion 4 Web Application Construction Kit* by Ben Forta. Published by permission of Macmillan Publishers Ltd. and the author. Chapter 11 appeared in two parts in the February and March issues of *ColdFusion Developer's Journal*. Parts 1 and 2 of Chapter 12 appeared in April and May. Part 1 of this article appeared in July. This article concludes the series.


```
<CFIF IsDefined("EmployeeID") >
```

This code sets a variable called `NewRecord` to Yes or No based on its existence. The following code sets `NewRecord` to No because the `IsDefined("EmployeeID")` test returned `TRUE`.

```
<CFSET NewRecord = "No">
```

If `IsDefined("EmployeeID")` returns `FALSE`, the following code sets `NewRecord` to Yes:

```
<CFSET NewRecord = "Yes">
```

Either way, once the first five lines of the template have been processed, you'll have a new variable called `NewRecord`, which indicates whether a new record is being added. You can then use this variable throughout the template wherever different code is needed for insertions or updates. Of course, you could have named this variable with some other name as well; the actual variable name is not that important, as long as it is descriptive.

Note: ColdFusion variables are special fields that you can create at any time; they can contain any values. Once a variable is created during the processing of a template, it is available for use until that processing is complete. Variables are assigned using the `CFSET` tag and can be reassigned using that same tag.

The `<CFQUERY>` that retrieves the record of the employee you want updated is conditional. It would make no sense to try to retrieve a record that does not yet exist in the table. Therefore, the entire `<CFQUERY>` statement is enclosed in a `<CFIF>` statement. The code `<CFIF NewRecord IS "No">` ensures that everything until the matching `</CFIF>` is processed only if this is an update.

The page title is also conditional; that way it accurately reflects the operation that is being performed, and if the operation is an update, the name of the employee being updated is displayed. Of course, displaying the employee name requires displaying dynamic data; the title is therefore enclosed within a `<CFOUTPUT>` block if the operation is an update:

```
<CFOUTPUT QUERY="Employee">
<TITLE>Update an Employee - #LastName#,
FirstName#</TITLE>
```

```
</CFOUTPUT>
```

The very first field in the form itself is a hidden field. The following code creates a hidden field containing the primary key of the record to be updated:

```
<INPUT TYPE="hidden" NAME="EmployeeID"
VALUE="#EmployeeID#">
```

This is required for the `<CFUPDATE>` tag to work, as explained earlier in this chapter.

This hidden field is only wanted if the operation is an update. A new employee ID is generated automatically at the time of data insertion for insert operations, so the entire `<INPUT>` tag is conditional and is only processed if the `<CFIF NewRecord IS "No">` condition returns `TRUE`. If `NewRecord` is Yes, all the code until the matching `</CFIF>` tag is ignored.

Conditional INPUT Fields

Next comes all the fields themselves, starting with the `FirstName` field. When adding a new record, the `FirstName` input field needs to look like this:

```
<INPUT TYPE="text" NAME="FirstName"
SIZE="30" MAXLENGTH="30">
```

When updating a record, the same field needs one additional attribute: `VALUE`. The parameter passed to the `VALUE` attribute is the `FirstName` column as retrieved by the `<CFQUERY>` tag. The complete field for an update operation, therefore, looks like this:

```
<INPUT TYPE="text" NAME="FirstName"
SIZE="30" MAXLENGTH="30"
VALUE="#Trim(FirstName)#">
```

Because the only difference between the two is the `VALUE` attribute, you can also make that conditional. This typically involves breaking the `<INPUT>` field over multiple lines, but that is allowed. The basic `<INPUT>` tag is first defined in the following code, but no terminating `>` is provided yet. Instead, you test to see if this is an update; if yes, the `VALUE` attribute is included within a `<CFOUTPUT>` block so that it can be populated with the current value. The terminating `>` appears after the condition, ensuring that the `VALUE` attribute will be contained within the `<INPUT>` tag if it is needed.

inteliant

www.inteliant.com

```

<INPUT TYPE="text" NAME="FirstName"
SIZE="30" MAXLENGTH="30"
<CFIF NewRecord IS "No">
<CFOUTPUT
QUERY="Employee">VALUE="#Trim(FirstName)"#</CFOUTPUT>
</CFIF>
>

```

That's all you need to create a conditional `<INPUT>` tag. The tag is a little more complicated to read – and

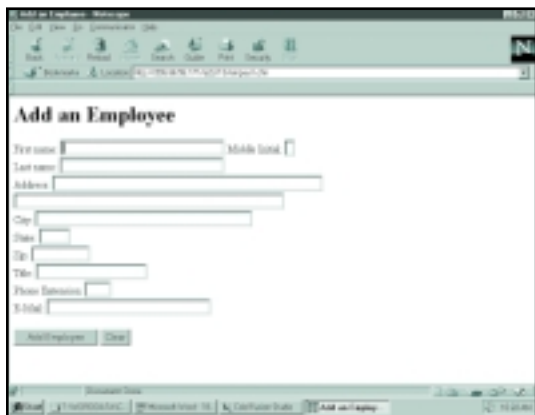


FIGURE 5: Templates can be reused when using conditional code; this Add form is also an Update form.

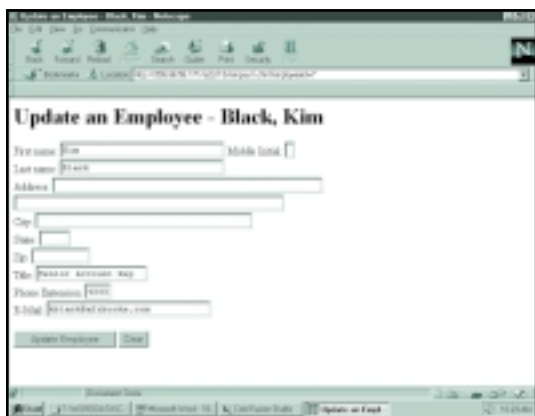


FIGURE 6: Templates can be reused when using conditional code; this Update form is also an Add form.

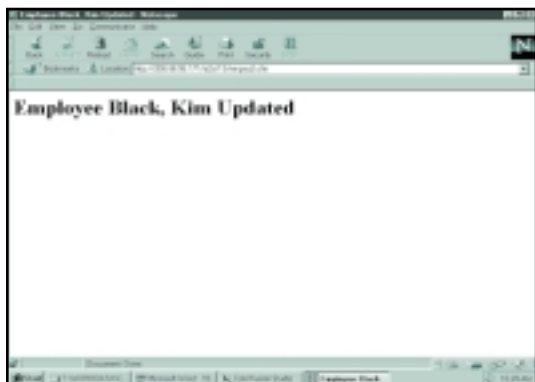


FIGURE 7: All the elements in a page can be created dynamically using ColdFusion variables and fields, including the HTML `<TITLE>`.

the generated HTML source code will likely contain multiple lines and blank lines in the middle of the tag – but the benefit here is that you'd only have to make any necessary tag attributes changes once. Similarly, if you needed to add input fields, you'd only have to make changes to a single template.

The final conditional code in Listing 11 is the submit button. Again, check the value of the `NewRecord` variable so you can specify an appropriate value for the submit button text.

You can now try this form. If you specify an `EmployeeID` parameter such as `?EmployeeID=7` in the URL, you'll be presented with an update form. Executing the same template without an `EmployeeID` parameter displays an add form. Both forms are shown in Figures 5 and 6.

But don't submit the form yet. Now you have to create a template that can conditionally perform the actual insert or update.

Processing Conditional Forms

Processing a conditional form requires that the destination template ascertain which operation needs to be performed, and there are many ways to do this:

- Embedding a hidden field in the form that specifies the operation
- Checking for the existence of a specific field, or the lack thereof (similar to what you did in Listing 11)
- Checking the value of a known entity (e.g., the submit button), that could have a different value based on the operation being performed

For this example you do the same thing you did in the form template itself – check for the existence of an `EmployeeID`. The hidden `EmployeeID`

field is only present if the operation is an update.

The conditional insert and update template is shown in Listing 12. Just as in Listing 11's form, the first thing you do is check for the presence of the `EmployeeID` field, but this time we explicitly check for an `EmployeeID` field within a FORM. The primary key form field must be present for an update to work. Chances are that there would not be any other `EmployeeID` field present, but just to make sure, you preface the field name with the FORM identifier, as follows:

```
<CFIF IsDefined("FORM.EmployeeID")>
```

You execute a `<CFUPDATE>` if the `EmployeeID` exists; otherwise you execute a `<CFINSERT>` – it's that simple. You now have a single template that can both insert and update employee records.

Unlike Listing 11, here you did not set a variable to indicate which operation to perform. Why not? There were many conditional elements within the code in Listing 11, and so you don't have to repeatedly check for parameter existence, you created a variable you could check instead. There is only one conditional code block here, and so you might as well perform the insert or update operations right there, within the conditional block.

The other thing you did in the conditional code is set a variable called `Operation`, which is set to `Inserted` and `Updated`. You then used this variable twice later, in the title and in the body. This way you did not need to create two more conditional code blocks. The variable contains the appropriate text to be displayed automatically, wherever it is used.

Try to update `EmployeeID 7`. Your browser should look like the one shown in Figure 7.

Note: There are an unlimited number of ways to structure your conditional code, and no single approach is right or wrong. The examples in this chapter demonstrate several different techniques, and you will undoubtedly develop several of your own. The only rule to remember is to make your code readable, manageable, and wherever possible, reusable.

Additional Code Reuse Techniques

While we're on the subject of code reuse, look at another useful way to

CAUTION

In order to submit the value of a button you must name the field with the `<INPUT>` NAME attribute. This way the browser can submit a name=value pair for the submit button.

Some older browsers do not support naming submit buttons, and they might ignore the attribute altogether.

eprise

www.eprise.com

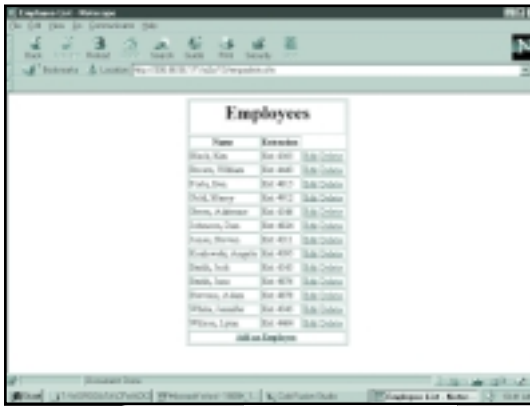


FIGURE 8: The employee administration page is used to add, edit, and delete employee records.

create reusable forms.

The combination Insert and Update form you created in Listing 11 works very well. The only problem with it is one of manageability. With so many <CFIF> statements you run the risk of introducing mismatched tags or typos, especially if you were to modify the form at a later date.

As a rule, programmers like to keep all conditional code in one place, creating a simpler program flow. The code in Listing 13 shows how to do this.

This template first determines whether this is an insert or an update operation. If it is an update, a <CFQUERY> is used to retrieve the current values. The fields retrieved by that <CFQUERY> are saved into local variables using multiple <CFSET> tags. There's no <CFQUERY> used if it is an insert operation, but <CFSET> is used to create empty variables.

Now look at the <INPUT> fields themselves. You'll notice that there is no conditional code within them as there was before. Instead, every

<INPUT> tag has a VALUE attribute regardless of whether this is an insert or an update. The value in the VALUE attribute is a ColdFusion variable, a variable that is set at the top of the template, not a database field.

Regardless of the operation, a set of variables will exist once that conditional code has been processed. The variables are empty if the operation is an insert, although they could also contain default values. If it is an update operation, the variables will contain the current values. Either way, there is a valid set of variables to work with.

If an insert form is being displayed, the FirstName variable is empty. (The variable would exist, it would just have "" as its value). The following code:

```
<INPUT TYPE="text" NAME="FirstName"
SIZE="30" MAXLENGTH="30" VALUE="#FirstName#">
```

translates into this:

```
<INPUT TYPE="text" NAME="FirstName"
SIZE="30" MAXLENGTH="30" VALUE="">
```

If an update form is being displayed for EmployeeID 7, the FirstName variable would contain the text Kim. The following code:

```
<INPUT TYPE="text" NAME="FirstName"
SIZE="30" MAXLENGTH="30" VALUE="#FirstName#">
```

translates into this:

```
<INPUT TYPE="text" NAME="FirstName"
SIZE="30" MAXLENGTH="30" VALUE="Kim">
```

The rest of the code in the template

uses these variables, without needing any conditional processing. Even the page title and submit button text can be initialized in variables this way, so <CFIF> tags are not needed for them, either.

Obviously this is a far more elegant and manageable form than the one you saw before. You may use either technique or a combination thereof – whatever suits you and your application.

Creating a Complete Application

Now that you've created add, modify, and delete templates, put it all together and create a finished application.

The following templates are a combination of all that you have learned in both this and the previous chapter.

The template shown in Listing 14 is the main employee administration page. It displays all the employees in the Employees table and provides links to edit and delete them (using the data drill-down techniques discussed in the previous chapter); the links also enable you to add a new employee. The administration page is shown in Figure 8.

Listing 15 is essentially the same reusable Add and Update form you created earlier. The only significant change is that the <FORM> ACTION has been changed so that template EMPAU5.CFM processes the responses.

Listings 16 and 17 perform the actual data insertions, updates, and deletions. The big change in these templates is that they themselves provide no user feedback at all. Instead, they return to the administration screen using the <CFLOCATION> tag as soon as they finish processing the database changes. <CFLOCATION> is used to switch from the current template being processed to any other URL, including another ColdFusion template. The following example code instructs ColdFusion to switch to the EMPADMIN.CFM template.

```
<CFLOCATION URL="empadmin.cfm">
```

This way, the updated employee list is displayed ready for further processing as soon as any change is completed.

Using CFIF to Create Conditional Code

The conditions you have created so far have all tested for equality or inequality; they determine if a field or value equals another value or does not

Operator	Alternate	Description
IS	EQUAL, EQ	Check that the right value is equal to the left value.
IS NOT	NOT EQUAL, NEQ	Check that the right value is not equal to the left value.
CONTAINS		Check that the right value is contained within the left value.
DOES NOT CONTAIN		Check that the right value is not contained within the left value.
GREATER THAN	GT	Check that the left value is greater than the right value.
LESS THAN	LT	Check that the left value is less than the right value.
GREATER THAN OR EQUAL	GTE	Check that the left value is greater than or equal to the right value.
LESS THAN OR EQUAL	LTE	Check that the left value is less than or equal to the right value.

TABLE 1: ColdFusion Conditional Operators

allaire

www.alliare.com

Operator	Description
AND	Conjunction; returns TRUE only if both expressions are true.
OR	Disjunction; returns TRUE if either expression is true.
NOT	Negation.

TABLE 2: ColdFusion Boolean Operators

equal it – but you are not limited to testing for equality. ColdFusion provides a complete set of conditional operators, and you can also combine conditions with AND and OR operators.

The complete list of operators is shown in Table 1. Many of the operators have alternate syntax, and you may use whatever syntax you are comfortable with. The syntax GREATER THAN OR EQUAL TO is very descriptive, but is also very wordy and takes up additional screen space, which might force you to have to scroll or wrap text over multiple lines. The abbreviated syntax GTE accomplishes the exact same thing and takes far less space, but is also less intuitive.

The Boolean operators available to you are shown in Table 2.

When combining conditions, each condition must be contained within a set of parentheses. The following example checks to see if both the FirstName and LastName fields exist:

```
<CFIF (IsDefined("FirstName")) AND
(IsDefined("LastName"))>
```

To check for either a first name or a last name, you could use the following:

```
<CFIF (IsDefined("FirstName")) OR (IsDe-
fined("LastName"))>
```

Often you will want to verify that a field is not empty and that it does not contain blank spaces. The following condition demonstrates how this can be accomplished:

```
<CFIF Trim(LastName) IS NOT "">
```


You can use the CONTAINS operator (used in these two examples) to check whether a value is within a range of values:

```
<CFIF "KY,MI,MN,OH,WI" CONTAINS State>
```

```
<CFIF TaxableStates CONTAINS State>
```

More complex expressions can be created by combining conditions within parentheses. For example, the following condition determines whether payment is by check or credit card; if payment is by credit card, it checks to ensure that there is an approval code:

```
<CFIF (PaymentType IS "Check") OR
((PaymentType IS "Credit")
AND (Approval Code IS NOT ""))>
```

As you can see, the ColdFusion conditional support is both extensive and powerful. 

Listing 10: EMPDEL1.CFM – Deleting Table Data with the SQL DELETE Statement

```
<CFIF IsDefined("EmployeeID") IS "No">
  Error! No EmployeeID was specified!
</CFABORT>
</CFIF>
```

```
<CFQUERY DATASOURCE="A2Z">
  DELETE FROM Employees
  WHERE EmployeeID = #EmployeeID#
</CFQUERY>
```

```
<HTML>
```

```
<HEAD>
<TITLE>Employee Deleted</TITLE>
</HEAD>
```

```
<BODY>
```

```
<H1>Employee Deleted</H1>
```

```
</BODY>
```

```
</HTML>
```

Listing 11: EMPAU1.CFM – Template That Displays an Employee Add or an Employee Update Form

```
<CFIF IsDefined("EmployeeID")>
  <CFSET NewRecord = "No">
<CFELSE>
  <CFSET NewRecord = "Yes">
</CFIF>
```

```
<CFIF NewRecord IS "No">
<CFQUERY DATASOURCE="A2Z" NAME="Employee">
  SELECT FirstName,
         MiddleInitial,
         LastName,
         Address1,
```

```
Address2,
City,
State,
Zip,
Title,
PhoneExtension,
Email
```

```
FROM Employees
WHERE EmployeeID = #EmployeeID#
</CFQUERY>
</CFIF>
```

```
<HTML>
```

```
<HEAD>
```

```
<CFIF NewRecord IS "Yes">
  <TITLE>Add an Employee</TITLE>
<CFELSE>
  <CFOUTPUT QUERY="Employee">
    <TITLE>Update an Employee - #LastName#, #FirstName#</TITLE>
  </CFOUTPUT>
</CFIF>
```

```
</HEAD>
```

```
<BODY>
```

```
<H1>
<CFIF NewRecord IS "Yes">
  Add an Employee
<CFELSE>
  <CFOUTPUT QUERY="Employee">
    Update an Employee - #LastName#, #FirstName#
  </CFOUTPUT>
</CFIF>
</H1>
```

```
<FORM ACTION="empau2.cfm" METHOD="POST">
```


house ad

```

<CFIF NewRecord IS "No">
  <CFOUTPUT QUERY="Empl oyee">
    <INPUT TYPE="hidden" NAME="Empl oyeeID" VALUE="#Empl oyeeID#">
  </CFOUTPUT>
</CFIF>

<P>

First name:
<INPUT TYPE="text" NAME="FirstName" SIZE="30" MAXLENGTH="30"
<CFIF NewRecord IS "No">
  <CFOUTPUT QUERY="Empl oyee">VALUE="#Trim(FirstName)#"</CFOUTPUT>
</CFIF>
>
Middle Initial:
<INPUT TYPE="text" NAME="MiddleInitial" SIZE="1" MAXLENGTH="1"
<CFIF NewRecord IS "No">
  <CFOUTPUT QUERY="Empl oyee">VALUE="#Trim(MiddleInitial)#"</CFOUTPUT>
</CFIF>
>
<BR>
Last name:
<INPUT TYPE="text" NAME="LastName" SIZE="30" MAXLENGTH="30"
<CFIF NewRecord IS "No">
  <CFOUTPUT QUERY="Empl oyee">VALUE="#Trim(LastName)#"</CFOUTPUT>
</CFIF>
>
<BR>
Address:
<INPUT TYPE="text" NAME="Address1" SIZE="50" MAXLENGTH="50"
<CFIF NewRecord IS "No">
  <CFOUTPUT QUERY="Empl oyee">VALUE="#Trim(Address1)#"</CFOUTPUT>
</CFIF>
>
<BR>
<INPUT TYPE="text" NAME="Address2" SIZE="50" MAXLENGTH="50"
<CFIF NewRecord IS "No">
  <CFOUTPUT QUERY="Empl oyee">VALUE="#Trim(Address2)#"</CFOUTPUT>
</CFIF>
>
<BR>
City:
<INPUT TYPE="text" NAME="Address1" SIZE="40" MAXLENGTH="40"
<CFIF NewRecord IS "No">
  <CFOUTPUT QUERY="Empl oyee">VALUE="#Trim(City)#"</CFOUTPUT>
</CFIF>
>
<BR>
State:
<INPUT TYPE="text" NAME="State" SIZE="5" MAXLENGTH="5"
<CFIF NewRecord IS "No">
  <CFOUTPUT QUERY="Empl oyee">VALUE="#Trim(State)#"</CFOUTPUT>
</CFIF>
>
<BR>
Zip:
<INPUT TYPE="text" NAME="Zip" SIZE="10" MAXLENGTH="10"
<CFIF NewRecord IS "No">
  <CFOUTPUT QUERY="Empl oyee">VALUE="#Trim(Zip)#"</CFOUTPUT>
</CFIF>
>
<BR>
Title:
<INPUT TYPE="text" NAME="Title" SIZE="20" MAXLENGTH="20"
<CFIF NewRecord IS "No">
  <CFOUTPUT QUERY="Empl oyee">VALUE="#Trim(Title)#"</CFOUTPUT>
</CFIF>
>
<BR>
Phone Extension:
<INPUT TYPE="text" NAME="PhoneExtension" SIZE="4" MAXLENGTH="4"
<CFIF NewRecord IS "No">
  <CFOUTPUT QUERY="Empl oyee">VALUE="#Trim(PhoneExtension)#"</CFOUTPUT>
</CFIF>
>
<BR>
E-Mail:

```

```

<INPUT TYPE="text" NAME="Email" SIZE="30" MAXLENGTH="30"
<CFIF NewRecord IS "No">
  <CFOUTPUT QUERY="Empl oyee">VALUE="#Trim(Email)#"</CFOUTPUT>
</CFIF>
>

<P>
<CFIF NewRecord IS "Yes">
  <INPUT TYPE="submit" VALUE="Add Empl oyee">
<CFELSE>
  <INPUT TYPE="submit" VALUE="Update Empl oyee">
</CFIF>
<INPUT TYPE="reset" VALUE="Clear">

</FORM>

</BODY>

</HTML>

```

Listing 12: EMPAU2.CFM – Template That Conditionally Inserts or Updates an Employee Record

```

<CFIF IsDefined("FORM.Empl oyeeID")>
  <CFSET Operation="Updated">
  <CFUPDATE DATASOURCE="A2Z" TABLENAME="Empl oyees">
<CFELSE>
  <CFSET Operation="Inserted">
  <CFINSERT DATASOURCE="A2Z" TABLENAME="Empl oyees">
</CFIF>

<CFOUTPUT>

<HTML>

<HEAD>
<TITLE>Employee #LastName#, #FirstName# #Operation#</TITLE>
</HEAD>

<BODY>
<H1>Employee #LastName#, #FirstName# #Operation#</H1>
</BODY>

<HTML>

</CFOUTPUT>

```

Listing 13: EMPAU3.CFM – Alternate Combination Insert and Update Form

```

<CFIF IsDefined("Empl oyeeID")>
  <CFSET NewRecord="No">
<CFELSE>
  <CFSET NewRecord="Yes">
</CFIF>

<CFIF NewRecord>
  <CFSET PageTitle = "Add an Employee">
  <CFSET ButtonText = "Add Empl oyee">
  <CFSET FirstName = "">
  <CFSET MiddleInitial = "">
  <CFSET LastName = "">
  <CFSET Address1 = "">
  <CFSET Address2 = "">
  <CFSET City = "">
  <CFSET State = "">
  <CFSET Zip = "">
  <CFSET Title = "">
  <CFSET PhoneExtension = "">
  <CFSET Email = "">
<CFELSE>
  <CFQUERY DATASOURCE="A2Z" NAME="Empl oyee">
    SELECT FirstName,
      MiddleInitial,
      LastName,
      Address1,
      Address2,
      City,
      State,
      Zip,
      Title,

```

```

PhoneExtension,
Email
FROM Employees
WHERE EmployeeID = #EmployeeID#
</CFQUERY>
<CFSET PageTitle =
"Update an Employee - " & Employee.LastName & ", " & Employee.FirstName>
<CFSET ButtonText = "Update Employee">
<CFSET FirstName = Trim(Employee.FirstName)>
<CFSET MiddleInitial = Trim(Employee.MiddleInitial)>
<CFSET LastName = Trim(Employee.LastName)>
<CFSET Address1 = Trim(Employee.Address1)>
<CFSET Address2 = Trim(Employee.Address2)>
<CFSET City = Trim(Employee.City)>
<CFSET State = Trim(Employee.State)>
<CFSET Zip = Trim(Employee.Zip)>
<CFSET Title = Trim(Employee.Title)>
<CFSET PhoneExtension = Trim(Employee.PhoneExtension)>
<CFSET Email = Trim(Employee.Email)>
</CFIF>

<CFOUTPUT>

<HTML>

<HEAD>
<TITLE>#PageTitle#</TITLE>
</HEAD>

<BODY>

<H1>#PageTitle#</H1>

<FORM ACTION="empau2.cfm" METHOD="POST">

<CFIF NewRecord IS "No">
  <INPUT TYPE="hidden" NAME="EmployeeID" VALUE="#EmployeeID#">
</CFIF>

```

```

<P>

First name:
<INPUT TYPE="text" NAME="FirstName" SIZE="30" MAXLENGTH="30"
VALUE="#FirstName#">
Middle Initial:
<INPUT TYPE="text" NAME="MiddleInitial" SIZE="1" MAXLENGTH="1" VALUE="#MiddleInitial#">
<BR>
Last name:
<INPUT TYPE="text" NAME="LastName" SIZE="30" MAXLENGTH="30" VALUE="#LastName#">
<BR>
Address:
<INPUT TYPE="text" NAME="Address1" SIZE="50" MAXLENGTH="50"
VALUE="#Address1#">
<BR>
<INPUT TYPE="text" NAME="Address2" SIZE="50" MAXLENGTH="50"
VALUE="#Address2#">
<BR>
City:
<INPUT TYPE="text" NAME="Address1" SIZE="40" MAXLENGTH="40"
VALUE="#City#">
<BR>
State:
<INPUT TYPE="text" NAME="State" SIZE="5" MAXLENGTH="5" VALUE="#State#">
<BR>
Zip:
<INPUT TYPE="text" NAME="Zip" SIZE="10" MAXLENGTH="10" VALUE="#Zip#">
<BR>
Title:
<INPUT TYPE="text" NAME="Title" SIZE="20" MAXLENGTH="20" VALUE="#Title#">
<BR>
Phone Extension:
<INPUT TYPE="text" NAME="PhoneExtension" SIZE="4" MAXLENGTH="4"
VALUE="#PhoneExtension#">
<BR>
E-Mail:
<INPUT TYPE="text" NAME="Email" SIZE="30" MAXLENGTH="30" VALUE="#Email#">

```

computerwork

www.computerwork.com

```
<P>
<INPUT TYPE="submit" VALUE="#ButtonText#">
<INPUT TYPE="reset" VALUE="Clear">
```

```
</FORM>
```

```
</BODY>
```

```
</HTML>
```

```
</CFOUTPUT>
```

Listing 14: EMPADMIN.CFM – Employee Administration Template

```
<CFQUERY DATASOURCE="A2Z" NAME="Employeees">
  SELECT FirstName, LastName, PhoneExtension, EmployeeID
  FROM Employeees
  ORDER BY LastName, FirstName
</CFQUERY>

<HTML>

<HEAD>
<TITLE>Employee List</TITLE>
</HEAD>

<BODY>

<CENTER>

<TABLE BORDER>
  <TR>
    <TH COLSPAN=3>
      <H1>Employeees</H1>
    </TH>
  </TR>
  <TR>
    <TH>
      Name
    </TH>
    <TH>
      Extension
    </TH>
  </TR>

  <CFOUTPUT QUERY="Employeees">
    <TR>
      <TD>
        #LastName#, #FirstName#
      </TD>
      <TD>
        Ext: #PhoneExtension#
      </TD>
      <TD>
        <A HREF="empau4.cfm?EmployeeID=#EmployeeID#">Edit</A>
        <A HREF="empdel2.cfm?EmployeeID=#EmployeeID#">Delete</A>
      </TD>
    </TR>
  </CFOUTPUT>

  <TR>
    <TH COLSPAN=3>
      <A HREF="empau3.cfm">Add an Employee</A>
    </TH>
  </TR>

</TABLE>

</CENTER>

</BODY>

</HTML>
```

Listing 15: EMPAU4.CFM – Employee Add and Update Form

```
<CFIF IsDefined("EmployeeID")>
  <CFSET NewRecord="No">
```

```
<CFELSE>
  <CFSET NewRecord="Yes">
</CFIF>
```

```
<CFIF NewRecord>
  <CFSET PageTitle = "Add an Employee">
  <CFSET ButtonText = "Add Employee">
  <CFSET FirstName = "">
  <CFSET MiddleInitial = "">
  <CFSET LastName = "">
  <CFSET Address1 = "">
  <CFSET Address2 = "">
  <CFSET City = "">
  <CFSET State = "">
  <CFSET Zip = "">
  <CFSET Title = "">
  <CFSET PhoneExtension = "">
  <CFSET Email = "">
<CFELSE>
```

```
<CFQUERY DATASOURCE="A2Z" NAME="Employee">
  SELECT FirstName,
    MiddleInitial,
    LastName,
    Address1,
    Address2,
    City,
    State,
    Zip,
    Title,
    PhoneExtension,
    Email
  FROM Employeees
  WHERE EmployeeID = #EmployeeID#
</CFQUERY>
<CFSET PageTitle =
  "Update an Employee - " & Employee.LastName & ", " & Employee.FirstName>
<CFSET ButtonText = "Update Employee">
<CFSET FirstName = Trim(Employee.FirstName)>
<CFSET MiddleInitial = Trim(Employee.MiddleInitial)>
<CFSET LastName = Trim(Employee.LastName)>
<CFSET Address1 = Trim(Employee.Address1)>
<CFSET Address2 = Trim(Employee.Address2)>
<CFSET City = Trim(Employee.City)>
<CFSET State = Trim(Employee.State)>
<CFSET Zip = Trim(Employee.Zip)>
<CFSET Title = Trim(Employee.Title)>
<CFSET PhoneExtension = Trim(Employee.PhoneExtension)>
<CFSET Email = Trim(Employee.Email)>
</CFIF>
```

```
<CFOUTPUT>
```

```
<HTML>
```

```
<HEAD>
<TITLE>#PageTitle#</TITLE>
</HEAD>
```

```
<BODY>
```

```
<H1>#PageTitle#</H1>
```

```
<FORM ACTION="empau5.cfm" METHOD="POST">
```

```
<CFIF NewRecord IS "No">
  <INPUT TYPE="hidden" NAME="EmployeeID" VALUE="#EmployeeID#">
</CFIF>
```

```
<P>
```

```
First name:
<INPUT TYPE="text" NAME="FirstName" SIZE="30" MAXLENGTH="30"
VALUE="#FirstName#">
Middle Initial:
<INPUT TYPE="text" NAME="MiddleInitial" SIZE="1" MAXLENGTH="1" VALUE="#MiddleInitial#">
<BR>
Last name:
```


Security Made Simple

Use CF's application framework to implement security across multiple templates



BY
KELLY
BROWN

One of the most common requests made of developers is that they create a password-protected area on a site, be it for registered users or administrative functions.

Ideally we'd like to have a system that automatically detects if users are logged in when they access a page and sends them to a login page if they're not. We also want to keep track of the users' state when they log in, so we can send them on to their original destination after the login process. Of course, this system would also be easy to implement with a minimal amount of effort on our part.

We can accomplish all these goals by using session variables and Allaire's application framework. Session variables allow us to check which users have logged in, while the framework gives us an easy way to implement security across multiple templates without additional work.

Logging In – The Simple Way

Let's start out with the basics: Why do I want to use the application framework file for security? For those not familiar with the application framework, the `application.cfm` is a special file that's run before any other ColdFusion template in the same directory or subdirectory. It's like cutting and pasting the code from the `application.cfm` into the top of each template. This model automatically provides security to all the ColdFusion templates in a directory without changing them. In addition, any new templates created in the directory are protected without any additional work.

How does this type of security work? CF provides a convenient method for tracking individual users called *sessions*. The session stays with the user and is used to

create variables that are accessible only to that user. For our security system we want to place an indicator in the user's session when he or she logs in. We can then check the indicator to see if the user is logged in. A good variable to use for this is the user's ID; it also comes in handy when we want to provide customized information for the user. If the user isn't logged in, we can redirect him or her to a login page. When we place this check in the `application.cfm`, it protects all the pages in its directory because the check is run before any other code in a template. A sample `application.cfm` might look like:

```
<CFIF NOT
IsDefined("Session.user_id")>
    <CFLOCATION URL="login.cfm">
</CFIF>
```

The `login.cfm` asks for the user's login name and password and then sets the user ID in the session variable. The next time the user tries to access a protected page after logging in, the session variable is defined and the code ends, allowing the rest of your template to execute.

Logging In – One-Stop Security

Hopefully you get the general idea and may have seen or used a similar system in other sites. However, we want more functionality from our security system. The first modification I make is to encode the login page within the `application.cfm`. This gives our model three advantages:

1. We don't have to track which page the user was trying to access.
2. All of the security procedures are

contained within a single file.

3. It allows us to pass on form variables.

When using this model you don't need to keep track of the templates the users were going to before they were required to log in. Why? Because they're still on the template they were trying to access. The `application.cfm` interrupts the normal functionality of a template and checks to see if users are logged in. If not, it displays a login form. The login form submits back to the template that they're on, which is again interrupted by the `application.cfm` that processes the login. If the login is okay, the normal functionality of the template is allowed to run.

At this point you're probably scratching your head, so here's a description of what you would see when using the model. You click on a link to the `report.cfm` template that has restricted access and aren't logged in. A form shows up asking for your login and password, but you see the `report.cfm` page in the URL. The `application.cfm` has interrupted the normal functionality of the `report.cfm` page and displayed the login form. Enter your login and password and you see the report you were expecting and you're still on the `report.cfm` page. The login form submitted back to the `report.cfm` page, which was interrupted by the `application.cfm` that processed the login and then ended, allowing the report to run normally. The next time you try to access the `report.cfm` you won't be prompted to log in because your session variable is now set.

With the login and login processing all in the application.cfm, we have a one-stop shopping security system. We drop the file in a directory, make a few modifications and we have security for all our templates. This system is very reusable; I have implemented it on several consulting engagements.

In the simple system we used a redirect to send users to the login page; when that happens we lose our ability to pass form variables and keep them as form variables. Using the improved system, we take all the form variables that were passed in the protected template and place them in hidden form fields along with the login name and password. When the login form is submitted, these variables are passed along with the form and can be accessed as the template expects them.

Code Walk-Through

The first thing we have to do in our application.cfm is to turn on the session variables – the foundation of our tracking system (see Listing 1). Then we check to see if you're logged in by checking if the user ID variable has been set in your session. If you're logged in, we skip the rest of the code and run the current page as normal. If you're not, the login code is executed.

At this point two things can be happening: either you're entering the page for the first time, in which

case we want to generate a login screen, or you've already submitted the login form. By checking to see if the form.login variable exists, we know if you're submitting the login form. If you're logging in, we check the database to verify your login name (your e-mail address) and password; You don't have to use a database to store user names and passwords; you could simply check for a predefined login name and password instead. If your login was valid we set your session.user_id variable. If it was invalid we set an error message. The message tells you only that your login was invalid; it could provide additional information, such as informing you that your login has expired.

The next step of the code checks to see that you're currently logging in or if an error message was generated while logging in. If neither of these is true, you've just completed a successful login and the code is skipped, allowing the rest of the page you're on to execute normally. If you're not logging in or received an error message when logging in, the login form is shown. This login page can easily be changed to match your site. We then check to see if we have an error message. If there's an error, it's displayed in bold red text. Next we generate the login form. For our form action we derive the name of the page we're currently viewing, causing the page to submit back to itself.

As part of the form action we also include any variables that we passed in the URL. Next we loop through the special Form.FieldNames list to create hidden form fields for any that we received. Then we display our login and password text boxes, which have been placed in an HTML table to align them nicely. After the page is displayed we use <CFABORT> to stop the template; this prevents the remainder of the page from being processed.

Potential Glitches

There are a few potential problems to look out for when using this model. The first is the possibility of a duplicate field name. The login form that's generated uses a login and password field. If you use the same form field name on one of the pages secured by the script, you'll have a conflict.

Another thing to watch out for are double quotes (the " character) in the form field data. The hidden form fields that pass along the user data use double quotes. When this happens you'll have problems with the HTML and the data won't be passed along correctly. You can eliminate this problem by using a regular expression to encode the double quotes.



ABOUT THE AUTHOR

Kelly Brown is the chief technology officer of About Web (www.aboutweb.com), an Internet solutions provider in the Washington, DC, metropolitan area. Kelly has a BS and MS in computer science and is a Microsoft certified system engineer.

K.BROWN@ABOUTWEB.COM

Listing 1

```
<!-- Turn On Session variables -->
<cfapplication name="AccessSecurity"
sessionmanagement="Yes"
setclientcookies="Yes"
sessiontimeout="#CreateTimeSpan(0, 2, 0, 0)#">

<!-- If not logged in, run login procedure -->
<CFIF NOT IsDefined("Session.user_id")>
<CFSET message="">
<!-- If submitting login form, process it -->
<CFIF IsDefined("Form.login")>
<!-- Check login and password -->
<cfquery name="check" datasource="users">
SELECT user_id
FROM users
WHERE email='#FORM.securitlogin#'
and password='#FORM.securitpassword#'
</cfquery>
```

```
<!-- If user found set session variable,
otherwise set error message -->
<CFIF check.RecordCount IS NOT 0>
<CFSET Session.user_id=check.user_id>
<CFELSE>
<CFSET message="Invalid Login.">
</CFIF>
</CFIF>

<!-- If logging in or invalid login
show login form -->
<CFIF NOT IsDefined("Form.login") or
message IS NOT "">
<html><head><title>User Login</title></head>
<body bgcolor="white">
<P align="CENTER"><B>Login</B></P>
<CFIF message IS NOT "">
<CFOUTPUT><P align="CENTER"><FONT color="red">
<B>#message#</B></font></P>
</CFOUTPUT><P>
```

```
<TD align=right><B>Password</B></TD>
<TD><input type="password"
        name="securitypassword"
        size=15></TD>
</TR>
</TABLE>
<P>
<input type=submit value="Login" name="Login">
</FORM>
</div>
</body>
</html>

<!-- Stop the template here when logging in,
      ignoring the rest of page -->
<CFABORT>
</CFF>

<!-- If our login was okay we fall through to the
      rest of the page -->
</CFF>
```

The code listing for
this article can also be located at
www.ColdFusionJournal.com

www.ectron.com

allaire

www.allaire.com

Introducing SMART OBJECTS

BY BENJAMIN PATE

Build extendable,
reusable, object-based
components using CFML

Developers have been working for some time to improve the way ColdFusion is used to develop Web sites. CF is a powerful presentation language for creating markup-based documents (HTML, WML and XML) with connectivity into component-based back-end systems such as C++, Java, COM/CORBA and more. With these add-ins, programmers have many ways to develop reusable object-oriented, component-based applications for the back end of a Web site. But there's still a hole in this architecture because we need a CFML component model for the Web site front end as well.



Many Web sites have interface requirements so similar to each other that it would be incredibly useful to have an object-based approach to code reuse for the Web site front end. I am sick and tired of writing the same application again and again just because one requirement was slightly modified. Imagine being able to encapsulate the interface for a company directory, content management system or departmental calendar in an easily understandable, extensible module. Imagine being able to customize its functionality for each individual Web site without having to vandalize the original code. I propose to fill this hole with an object-oriented component model for the Web site front end called *SmartObjects*. This article provides an overview of the SmartObjects model and demonstrates the value of object-based inheritance with a simple application that adds/updates/deletes from a database table.

Design Objectives

I'm familiar with other attempts to convert CF custom tags into reusable components but I haven't been sufficiently satisfied to commit to using any of them. Something is always missing in terms of ease of use, flexibility and extensibility. Because ColdFusion is a presentation language, it's difficult to create a static component that works seamlessly among multiple applications. If I want to reuse a component between two completely separate Web sites, I must try to create a generic interface that won't look out of place on any of them.

But then what do I do when one client asks for one set of additions to the design and another client has requirements that pull in an opposite direction? With existing component models I still end up having to fork the code – make different copies of the source code – which is asking for trouble in the long run. It's very difficult to maintain two sets of code in parallel. One Web site will usually lag behind in development, the two code bases will drift farther apart, and they'll eventually become completely unrelated. This is the primary problem addressed by SmartObjects through the use of a form of inheritance similar to true object-oriented languages. Instead of rewriting code or cutting and pasting the parts that are needed, programmers who use SmartObjects are able to inherit the functionality they need and override the parts they don't.

Additionally, I don't want to bury my component code in the Custom Tags directory. Depending on the environment in which code is developed, access to the Custom Tags directory can be anywhere

from inconvenient to downright impossible. Furthermore, components that are installed in the Custom Tags directory still need a stub in the Web root to call them. This method would lead to splintered application code scattered across the whole machine and can be difficult to debug. Instead, there needs to be a way to execute the component as a stand-alone application and there should be an easy way to locate the component and test it during development. This means that the component code must be located together in one place in the Web root like any other application.

Finally, I want to use a model that doesn't require me to change my development style. I've worked hard to create the development patterns that I follow and it doesn't make sense to have to throw them out just to start reusing components. The component model should provide services to the programmer without dictating design.

SmartObjects accomplishes all of these requirements without adding processing overhead to the system. Execution time is optimized with a sort of "just-in-time" (JIT) compiling that saves the complex work of creating object definitions in a cache of application variables. Since classes and objects have to be configured only once at start-up, the only overhead in using SmartObjects comes from the context switch to a custom tag and a <CFINCLUDE>. In terms of performance, this makes SmartObjects lean and mean.

Architecture

A complete background in object-oriented programming is well beyond the scope of this article. I must assume that the reader has a solid grasp of basic OO principles, but ask that you give me a little artistic license with the concept. SmartObjects is neither complete nor pure objects by any means, but it's a decent approximation...and it's a great leap forward in code reuse for CFML.

The SmartObjects model consists of three custom tags that provide a simple foundation for developing object-based

CF applications (see Table 1). They should be installed anywhere within the Custom Tags directory so that they're accessible to all CF applications. The tags can be downloaded from the Allaire Tag Gallery or from my Web site at www.smart-objects.com. Using the three custom tags, let's examine the anatomy of a single stand-alone class. I'll dive into the process of extending it in a moment.

In object-oriented languages, logical units of code are grouped into classes. Using SmartObjects is similar – each class comprises a group of CF templates in any directory. To turn a directory into a class, all you need to do is add a class definition file called public.cfm. The CF templates in the directory are then available as "methods" of the class. Listing 1 provides the public.cfm file for the table maintenance class. This file is processed only once, when the application is first used. After that the class definition is kept in a CF structure called Application.classes. This saves time in recalculating the locations of methods and makes the job of figuring out which template to include much easier. The class attribute must point to the location of the class inside the Web root or in a location that's been mapped through the ColdFusion administrator.

As with the <CFMODULE> tag, periods in the class name represent subdirectories in the file system. This notation preserves the "object-ness" of SmartObjects. The *optional inherit* attribute lists the name of one or more base classes that we'll extend – I'll explore this in more detail later. The *methods* attribute lists the names of the files that will be accessible through the object interface. Each method name matches a file name exactly, so that executing:

```
<CF_Call object="objectname" method="find">
```

includes the find.cfm file from the appropriate object directory. It isn't necessary to list all the files in the directory as class methods. Those you don't list won't be available to the <CF_Call> tag and are similar to private functions in C++.

Once created, SmartObjects classes can be called any way you want. They can

<CF_Class>	Class Declaration. Registers a class in the application server, compiles a list of methods that it supports and identifies any base classes that the class extends. Names and locations of these methods are stored in Application variables for quick access in the future.
<CF_Object>	Class constructor. Creates an instance of the class in the caller's namespace that can be accessed in the application.
<CF_Call>	Calls a class method. This custom tag uses a <CFINCLUDE> to execute the appropriate method template.

TABLE 1: The three custom tags you need

Listing 3	delete.cfm	Action Page. Deletes one row from the table and returns the user to the find page.
Listing 4	edit.cfm	Display Page. Allows user to edit record information. Posts variables to the Update method.
Listing 5	find.cfm	Display Page. Shows a simple listing of all items in the table. Provides a list of links that point to an edit form.
Listing 6	qry_AllRecords.cfm	Query Page. Returns the entire table in a CFML query.
Listing 7	qry_ThisRecord.cfm	Query Page. Dumps all of the fields for a specific record into a query.
Listing 8	update.cfm	Action Page. Uses <CFUPDATE> to update the table one record at a time.

TABLE 2: Descriptive key to Code Listings 3–8

be hidden inside an application and called as support functions. In order to operate as a stand-alone application, the example class in this article borrows several conventions from FuseBox. Each method expects to be called from an index.cfm file that determines the appropriate method to call using a URL variable. I've chosen to use URL.method only because it's simpler and more descriptive than URL.FuseAction, but it serves exactly the same purpose. Nothing about SmartObjects requires this method of developing applications but it can be a powerful way of developing reusable code.

Listing 2 provides the index.cfm file that instantiates the object and calls the appropriate method. This file doesn't need to be in the same directory as the public.cfm file or the rest of the class methods. When building a stand-alone class it's simplest to put it in the same directory because the entire application then resides in the same location. But remember that the idea is to reuse this code among many different applications, so the code that calls this class will eventually be distant from the class itself.

Calling <CF_Object> is simple and doesn't require many parameters. The class attribute identifies the name of the class being instantiated. Again, this class must include a public.cfm file and must be available to ColdFusion as an include file. This means that the template must reside in the Web root or within a directory that's been mapped with the ColdFusion Administrator. The object attribute provides the name of the structure variable in the caller's scope that contains the object and its properties. So, when I call:

```
<CF_Object class="myclass"
object="session.item">
```

the custom tag creates a new structure variable called session.item. Object properties variables are stored as keys/value

pairs in the structure along with a few values that are required by the system. I can then access and update properties for this variable the same as any other CF variable. These variables can be set using the <CF_Object> tag by including them as attributes of the tag, or they can be set afterwards by assigning the key/value pairs directly to the structure using <CFSET>. I'll describe how the methods access these variables in just a moment.

Once the object has been instantiated, the bottom of the index.cfm file calls the appropriate method using the <CF_Call> tag and displays the output inside the interface that you specify. Another interesting convention is demonstrated here. Sometimes it's useful for one method to include another to chain them together without calling a <CFLOCATION> between them. For example, an update script might first call the delete method to clean out old records before continuing with its operation. To do this, I set a Request.location variable at the end of a method that performs an action such as updating the database. Then the index.cfm template performs the <CFLOCATION> after the methods have been executed. This gives the calling application greater control over process flow and allows for more flexibility and modularity in programming style.

To execute a method, the <CF_Call> tag looks in a global data structure called Application.classes for the location-appropriate template and uses <CFINCLUDE> to include the file. This means that the method executes from within the <CF_Call> custom tag and has its own variable namespace that's separate from the template that invokes it. Thus all variables in SmartObjects methods are encapsulated from the caller through the same mechanism that protects any other custom tag.

The <CF_Call> tag also provides a way for the method to access the object properties stored in the caller's object variable. In each of the methods in Listings

3–8 you'll notice a reference to a variable called *this*. The *this* pointer is a structure variable that's created in the scope of the <CF_Call> custom tag that points back to the object variable in the caller's scope. Since ColdFusion structures are passed by reference instead of value, it's possible to create two structure variables in separate namespaces that point to the same data. When the root-level application instantiates an object named Request.feedback_form in its own scope, the methods of that object can reference that data structure by calling *this* in the local scope. The *this* pointer is useful because it means that methods can call other methods in the same object without having to know the specific name that the caller used for the object variable. It also provides a mechanism for methods to store their own local data structures.

Because there's no such thing as information hiding in ColdFusion, there's no way to create completely private methods or data using SmartObjects. There's no way to prevent another part of the application from accessing internal data. But this architecture does provide a logical way for components to keep their own information out of the way of the rest of the application. Just like objects created with Perl, it's up to the programmer to obey the conventions required by the individual object.

Example Application

Now that the foundation is laid and you have a basic understanding of the SmartObjects technique, we can begin to program the object. From here, Listings 3–8 provide the class methods of the demonstration application that perform the actual work of the class (see Table 2). This demonstration application provides simple table maintenance for a generic database table. Once you're comfortable with the convention of sending all links and form posts back to the index.cfm and controlling program flow with the URL.method variable, it's fairly straightforward ColdFusion. Nothing is too revolutionary, yet....

Extending the Base Class

By defining a directory of templates as a class, you're able to extend its abilities with other classes without disturbing the original code. Now that you have a base class to work with, you can modify it to fit a specific application or improve on it easily and in a modular fashion. Since the demonstration class is a general-purpose table maintenance object, the subclass can extend it by providing a customized interface for a spe-

allaire

www.allaire.com

cific table. This is accomplished by overriding the edit method with a new template. Everything else is included automatically from the parent class.

For an example of a SmartObjects subclass definition, look at Listing 9. This new directory (the subclass) extends our original demonstration application (the base class) by naming it in the inherit attribute of the <CF_Class> tag. Normally, when the <CF_Class> custom tag is executed, it populates the Application.classes structure with the list of supported methods and their locations on the server. If the inherit attribute is present, then <CF_Class> begins by initializing the base if necessary by including its class definition template (i.e., the public.cfm template in that directory). Then the custom tag combines the list of supported methods from the two classes so that subclass methods always take precedence over those of the base class. All of the methods of the base class will be present and will be overlaid with the methods of the subclass. When both parent class and subclass define a method with the same name, the method from the subclass will be used.

It's also possible to inherit methods from more than one class at a time by passing a comma-separated list of parent classes. In this case the first items in the list take precedence over later ones. Once all of the inherited methods have been assembled, the entire structure is overlaid with the methods of the subclass.

With the subclass definition complete, all of the functionality of the base class is now available to our new application and all we have to do is define those methods that are specific to the new application. Listing 10 provides a new index.cfm file that instantiates this new class. It's identical to the original save for the object that it instantiates. I could add anything else that the new application requires: more functionality, additional pages or different data sources. All of the services of the base class are at the disposal of the new application. This new class is designed to maintain a table of people. For simplicity, I'm only extending the edit method from the base class. The new component that presents a slicker customized interface is found in Listing 11.

I've used this technique quite effec-

“”
One programming
technique that has
become popular
is to move each
query into its own
separate file

tively to create a diverse hierarchy of classes that enable me to re-create large applications very quickly. A generic content management class is easily extended to fit the specific needs of an individual Web site and can draw on the services of a file upload class, an RTF to HTML conversion class and others if necessary. All of these can be extended with a custom written *display* function that draws the content to the screen using whatever format the current Web site demands.

As another example of SmartObjects inheritance, several applications can draw from the same base class and extend it in dramatically different ways. It's easy to create a base class that serializes itself to WDDX and writes itself to a disk and then deserializes itself back from the disk. This “serializable” class can be used as a base class to provide simple persistence to any number of unrelated objects.

All of these pieces can be plugged in to a complete Web application just as they could in C++, Java or any other object-oriented language. But now these pieces are created in the powerful presentation language of CFML.

For example, one programming technique that has become popular is to move each query into its own separate file. By defining queries separately as methods of a class, another subclass written in the future could override the query to pull its data from a completely different source, such as a text file, LDAP directory or a COM object without regard to the rest of the class. Or the price column of a prod-


uct query could be modified with specific pricing rules before being returned to the application. With this it is possible to change the operation of an entire application by updating just one template.

Though it may be possible for other methods to emulate this kind of operation using carefully crafted include files, the code would become increasingly complicated as the tree of parent and child classes grows. Each new application would have to manage the list of locations of its predecessors and must accommodate a tree of templates that grows in complexity with every new application. The magic is in the SmartObjects custom tags because they manage this automatically. SmartObjects classes actually become simpler as the tree expands because the only code that needs to be added is for the templates that are unique to each new class.

Exploration

There's plenty more beneath the surface of SmartObjects. It serves only as the foundation on top of which many other programming techniques and styles can be laid. The design has limitless possibilities.

As with most demonstrations, the example presented here is simple so the technique is easier to identify. There are many industrial-strength applications of SmartObjects code in existence today. For example, I've created a customer survey application that can be customized rapidly for any site that requires it, and a portable authentication applet to secure Web site access. I have also completed a set of classes for a content management application that can easily be molded to fit the specific needs of many diverse Web sites.

I'm looking forward to the new opportunities for real component-based code reuse that SmartObjects provides. With it, there exists a complete architecture for object-based programming at every level of Web site development. I've already seen the benefits of simplifying code maintenance by using this technique. These benefits increase exponentially with each new class. I hope you find SmartObjects as useful as I have. 

About the Author

Ben Pate has been a Web developer for five years and holds a bachelor's degree in computer science from UCLA. Focusing on creating Web applications with ColdFusion, he has served as a consultant and contractor for a wide range of organizations from small- and medium-sized companies to the Fortune 500. He lives in Redondo Beach, California.

benjamin@pate.org

The SmartObjects model consists of three custom tags that provide a simple foundation for developing object-based CF applications (see Table 1). They should be installed anywhere within the Custom Tags directory so that they're accessible to all CF applications. The tags can be downloaded from the Allaire Tag Gallery or from my Web site at www.smart-objects.com.

JDJSTORE.COM GUARANTEED! LOWEST PRICES!

Guaranteed Best Prices

JDJ Store Guarantees the Best Prices. If you see any of our products listed anywhere at a lower price, we'll match that price and still bring you the same quality service.

Terms of offer:

- Offer good through November 30, 2000
- Only applicable to pricing on current versions of software
- Offer does not apply towards errors in competitors' printed prices
- Subject to same terms and conditions

Prices subject to change.
Not responsible for typographical errors.

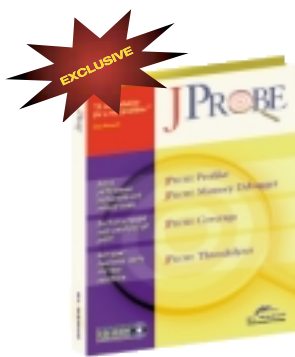
Attention Java Vendors:

To include your product in JDJStore.com, please contact amanda@sys-con.com

GALILEO DEVELOPMENT SYSTEMS

Intr@Vision Foundation

JDJStore.com **\$3499⁹⁹**



KL GROUP'S

JProbe Developer Suite v2.8 with Gold Support

JDJStore.com **\$1498⁹⁹**



ALLAIRE

ColdFusion Studio v4.5

JDJStore.com . . . **\$462⁹⁹**

ALLAIRE

HomeSite 4.5

HomeSite 4.5 from Allaire HomeSite is the award-winning HTML editing tool that lets you build great Web sites in less time, while maintaining pure HTML. Created by Web developers for Web developers, only HomeSite gives you precise layout control and total design flexibility, while delivering the latest Web technologies.



HomeSite 4.5 **\$93⁹⁹**

CLOUDSCAPE INC.

Cloudscape Single User Developer License

Building on its technology lead as the industry's first embeddable Java database designed for distributed, off-line, and mobile computing, Cloudscape Release 2.0 is designed to support eBusiness applications such as "smart" eCatalogs and supply chains. Enhancements in 2.0 include:

*Multi-user concurrency that supports hundreds of users. *Advanced security that enables only authorized applications to read data. *Substantially faster load of initial information into the database Cloudscape 2.0 is bundled with Cloudview (a schema browser utility). Includes one year of support from the Cloudscape services organization.



Cloudscape Single User Developer License **\$894⁹⁹**

ADOBE

Adobe Illustrator v9.0

Adobe Illustrator 9.0 software puts the power of editable vector graphics to work for the Web. Plus it expands your creative range and enhances your productivity with unlimited transparency capabilities, powerful object and layer effects, and other innovative features. And its tight integration with other Adobe software ensures a smooth, efficient workflow. Showcase your graphics in print, on the Web, or in dynamic media with complete creative freedom.



Adobe Illustrator v9.0 CD Mac **\$398⁹⁹**

PROTOVIEW

JSuite v2.5

JSuite, a low priced bundle of four of the industry's leading JavaBeans component products. It includes: CalendarJ: Calendar Display Component. DataTableJ: The Fastest Grid Component On The Net! TreeViewJ: Feature-Rich TreeView Component. WinJ Component Library: A Series Of UI Components.



JSuite v2.5 **\$794⁹⁹**

ALLAIRE

JRun Server v3.0 Enterprise

JRun 3.0 is an easy-to-use J2EE application server and integrated development environment for building and deploying server-side Java applications. From e-commerce to business automation, JRun is the easiest way for developers to deliver advanced business systems faster and at a lower cost than you'd ever thought possible.



JRun Server v3.0 Enterprise 2 CPU Licenses . . . **\$8602⁹⁹**

eHELP

RoboHELP Office 2000

RoboHELP Office provides a user-friendly WYSIWYG authoring environment for creating JavaHelp. RoboHELP guides you through the process so you can create a great JavaHelp system - with point-and-click and drag-and-drop ease. Now you can create JavaHelp systems as easily as you create WinHelp, Microsoft HTML Help, and WebHelp (cross platform Help) from the same source product - all with RoboHELP Office.



RoboHELP Office 2000 **\$898⁹⁹**

Listing 1: Public.cfm

```
<!-- Parent class definition --->
<CF_Class
  class="object-demo1"
  methods="find, edit, update, delete, qry_ThisRecord, qry_AllRecords"
>
```

Listing 2: Index.cfm

```
<cfapplication name="SmartObjects demo">

<cfparam name="method" default="find">

<!-- Instantiate the object --->
<CF_Object class="object-demo1" object="obj">

<!-- Define object defaults --->
<cfset obj.table="anytable">
<cfset obj.id="id">
<cfset obj.datasource="odbc-demo">

<!-- Output the page --->
<cfheader name="Expires" value="#Now()#">
Top Navigation Goes Here
<hr>

<cfset Request.Location = "">
<CF_Call object="obj" method="#method#">

<hr>
Bottom Navigation Goes Here

<!-- Follow up with another page if appropriate --->
<cfif Len(Request.Location)>
  <cflocation url="#Request.Location#">
</cfif>
```

Listing 3: Delete.cfm

```
<CFQUERY datasource="#this.datasource#">
  DELETE FROM #this.table#
  WHERE #this.id# = #id#
</cfquery>

<cfset Request.Location = "#CGI.SCRIPT_NAME#?method=find">
```

Listing 4: Edit.cfm --->

```
<!-- Call method to get this record from the database --->
<CF_Call object="this" method="qry_ThisRecord">

<cfoutput>

  <!-- Redirect form post back to the same template that called this
  method --->
  <!-- but tell it to call the update method --->
  <form action="#CGI.SCRIPT_NAME#?method=update&id=#Evaluate(this.id)#"
  method="post">

    <!-- Simple dynamic form loops over all columns in table --->
    <table>
      <CFLOOP index="i" list="#Request.qry_ThisRecord.ColumnList#">
        <tr>
          <td>#i#</td>
          <td><input name="#i#" value="#Evaluate("Request.qry_ThisRecord.
          &i)##"></td>
        </tr>
      </cfloop>
    </table>
    <p>
      <input type="submit" value="Update Now">
      <input type="button" value="Return to Index" onclick="window.docu-
      ment.location='#CGI.SCRIPT_NAME#?method=find'">
      <input type="button" value="Delete" onclick="window.document.loca-
      tion='#CGI.SCRIPT_NAME#?method=delete&id=#id#">
    </form>

  </cfoutput>
```

Listing 5: Find.cfm

```
<!-- Get all records from the database --->
<CF_Call object="this" method="qry_AllRecords">

<!-- Print out a simple list of items in this table --->
```

```
<cfoutput query="Request.qry_AllRecords">
  <a
  href="#cgi.script_name#?method=edit&id=#Evaluate(this.id)#">#Name#</a>
<br>
</cfoutput>
```

Listing 6: qry_AllRecords.cfm

```
<!-- Ensure that this query is not executed more than once per
request --->
<CFIF NOT IsDefined("Request.qry_AllRecords")>

  <!-- Query all data from the table. --->
  <!-- The #this.table# variable is defined by the template --->
  <!-- that instantiates the object (index.cfm) --->
  <cfquery datasource="#this.datasource#" name="Request.qry_All-
  Records">
    SELECT *
    FROM #this.table#
  </cfquery>

</cfif>
```

Listing 7: qry_ThisRecord.cfm

```
<!-- Ensure that this query is not executed more than once per
request --->
<CFIF NOT IsDefined("Request.qry_ThisRecord")>

  <!-- Declare required parameter to prevent application from choking
  --->
  <cfparam name="#this.id#" default="0">

  <!-- This dynamic query uses local object variables to determine --
  -->
  <!-- which table and record to fetch data from --->
  <cfquery datasource="#this.datasource#" name="Request.qry_This-
  Record">
    SELECT *
    FROM #this.table#
    WHERE #this.id# = #Evaluate(this.id)#
  </cfquery>

</cfif>
```

Listing 8: Update.cfm --->

```
<!-- Update the table --->
<cfupdate datasource="#this.datasource#" tablename="#this.table#">

<!-- Set the location of the next template to execute --->
<cfset Request.Location = "#CGI.SCRIPT_NAME#?method=edit&id=#Eval u-
ate(this.id)#">
```

Listing 9: Public.cfm --->

```
<!-- Subclass definition --->
<CF_Class
  class="object-demo2"
  inherits="object-demo1"
  methods="edit">
```

Listing 10: Index.cfm --->

```
<cfapplication name="SmartObjects demo2">

<cfparam name="method" default="find">
<cfset Application.classes = StructNew()>

<!-- Instantiate the object --->
<CF_Object class="object-demo2" object="obj2">
<!--
<cfloop item="i" collection="#Application.classes['object-demo2']#">
  <cfoutput>
    #i# = #Application.classes['object-demo2'][i]#<br>
  </cfoutput>
</cfloop>

<cfabort>
--->
<!-- Define object defaults --->
<cfset obj2.table="anytable">
<cfset obj2.id="id">
<cfset obj2.datasource="odbc-demo">

<!-- Output the page --->
<cfheader name="Expires" value="#Now()#">
```

```
<!-- Follow up with another page if appropriate -->
```

Listing 11: Edit.cfm

```

</tr>
<tr>
<td>Ci ty</td>
<td>&nbsp; <input name="Ci ty"
val ue="#Request. qry_Thi sRecord. Ci ty#" type="text" si ze="30"</td>
</tr>
<tr>
<td>State</td>
<td>
<table cel l paddi ng="0" cel l spac i ng="0">
<tr>
<td>&nbsp; <input name="State"
val ue="#Request. qry_Thi sRecord. State#" type="text" si ze="2"
maxl ength="2" si ze="30"></td>
<td>&nbsp; &nbsp; &nbsp; <input type="button" val ue="Update Now"
</td>
</tr>
</td>
</tr>
</table>
<p>
<input type="submit" val ue="Update Now">
<input type="button" val ue="Return to Index" oncl ick="wi ndow. docu-
ment. l ocat i on=' #CGI . SCRIP T_NAME#?method=fi nd' ">
<input type="button" val ue="Del ete" oncl ick="wi ndow. document. l oca-
ti on=' #CGI . SCRIP T_NAME#?method=del ete&i d=#i d#' ">
</p>
</form>
</cfoutput>

```

The code listing for
this article can also be located at
www.ColdFusionJournal.com

www.virtualscape.com

Extending CFFORM with Customized JavaScript Validation

Write your own functions to perform additional validation on your CFFORM tags

BY SELENE BAINUM

Anyone who's needed JavaScript validation for his or her forms knows how easy it is to use ColdFusion's CFFORM tag. It's a quick and easy way to ensure that the form is filled out properly. However, CFFORM can be as limited as it is useful (see Figure 1).

Several types of validation that aren't available are:

- Making a single-select box required
- Making checkboxes and radio buttons required

- Limiting the number of checkboxes or multiselect options selected
- Restricting dates entered to a certain range.

Many developers scrap CFFORM altogether and just use the HTML FORM tag in conjunction with their own JavaScript validation. Rarely do you hear about multiple customized validation scripts being used with the CFFORM tag.

It's possible to use both – good news for

anyone who already uses CFFORM and wants to add on to it or for those who're tired of writing JavaScript to ensure basic validation.

Modular Validation Scripts

The CFFORM JavaScript validation is modular. There's a function for each type of validation. These functions are called by the main function for each form field that uses them. The validation to make sure a text box is filled will be called five times if you have 5 CFINPUT TYPE = "text" fields with REQUIRED set to yes. Our customized validation will be set up the same way.

First you need to create your SCRIPT tag. Place your <SCRIPT> tag inside the <HEAD> tags. ColdFusion positions the CFFORM JavaScript just above the </HEAD> tag. If your JavaScript isn't inside the <HEAD>, it'll be below the CFFORM JavaScript. It'll still work, but it's harder to debug your JavaScript when it's at line number 400 in your source code. It's much easier to have it at the top of your source. Your customized validation will be run after the CFFORM validation completes successfully.

```
<HTML>
<HEAD>
<TITLE>Page Title</TITLE>
<SCRIPT LANGUAGE = "JavaScript">
function ExtendJS(FormName) {
...
}
</SCRIPT>
</HEAD>
```

Your JavaScript will contain several functions. The one that processes when the form is submitted is the one that ultimately decides whether the form should indeed be submitted or if the action should be halted and an alert box returned to the user. This function will call your validation functions. You'll have one validation function for each type of validation you wish to perform. We're using ExtendJS() as the main function. Inside the parentheses after the function is the variable FormName. To make the code modular we're also passing in the form's name to the function; it will be represented in the functions as the FormName variable.

Calling Your Scripts

For this function to be processed when the form is submitted, you must add an onSubmit method to the CFFORM tag:

```
<CFFORM NAME = "Extend" ACTION =
"Action.cfm" onSubmit = "return
ExtendJS('Extend')">
```


In your function call you need to include the name of your FORM. This is the value of the NAME attribute of the CFFORM tag. In the example above the name of the form is Extend.

Now we can start adding validation functions and then calling them from our main function.

Two of the most common validations CFFORM lacks are making a single-select box required (really required, not just passing the first value) and making a text area field required.

Required Select Boxes

Unless an option is set to be selected when the page loads, the first item in a single-select box is selected by default. You must have a value selected in a single-select box. When using a CFSELECT populated by a query, the first value from the query will be selected automatically. If the user doesn't make any changes, that first value will be passed to the action page. There's no way to make sure the user actually chooses an item. To really make a single-select box required you must use the HTML SELECT tag and then have the options generated inside a CFOUTPUT tag. Before that CFOUTPUT tag, however,

include an empty option tag. This empty option is what will be selected by default:

```
<SELECT NAME = "SelectBox">
  <OPTION VALUE = "">Select One
  <CFOUTPUT QUERY = "MyQuery">
    <OPTION VALUE =
      "#Field#">#Field#
  </CFOUTPUT>
</SELECT>
```

You then need a function to ensure that a value is actually selected by the user. The first option in a select box has an index of 0. If that option is the one selected, you know the user hasn't really selected an option. In that case the function will return a value of false. If any other option is selected, the function will return a value of true. This function can be placed inside the same SCRIPT tag as your main function (ExtendJS), but it must be underneath or below it, not inside it.

function

```
SingleSelectRequired(Form, Field) {
  var itemSelected =
    eval("document." + Form + "." +
      Field + ".selectedIndex");
  if (itemSelected == 0) {
    return false;
  } else {
    return true;
  }
}
```

To use this validation for your select boxes you'll need to call the validation function from the ExtendJS function:

```
function ExtendJS(FormName) {
  if (!SingleSelectRequired(FormName,
    'SelectBox')) {
    alert("You must select an
      item from the drop-down
      list.");
    return false;
  }
  ...
}
```

In the script above you're calling the SingleSelectRequired function and passing it the name of the form and the select box. To use the SingleSelectRequired function for multiple select boxes, call the function for each one and pass it the appropriate field name. If the function returns false, the alert message will display for the user and the form won't submit. If the function returns true, the script will continue running. If all validations pass, the form will be submitted.

Required Textarea Fields

One of the most commonly used form fields, TEXTAREA, doesn't even have a CF equivalent. This makes validating a TEXTAREA very difficult. To include a TEXTAREA in your validation, first add the field to your form:

```
<TEXTAREA NAME = "Comments" COLS
  = "30" ROWS = "4" WRAP =
  "virtual"></TEXTAREA>
```

Now add a validation function:

```
function TextAreaRequired(Form,
  Field) {
  var length = eval("document." +
    Form + "." + Field +
    ".value.length");
  if (length == 0) {
    return false;
  } else {
    return true;
  }
}
```

This validation function is slightly different than the one for a text box. Instead of just checking to see if the form field has a value, see if the form field's value has a length. If the length is 0, you know nothing was entered into the TEXTAREA. To call this function, add another call to your ExtendJS function:

```
if (!TextAreaRequired(FormName,
  'Comments')) {
  alert("You must enter some
    comments.");
  return false;
}
```

Additional Tips

The two examples above show how to validate HTML FORM fields inside a CFFORM. You can also write your own functions to perform additional validation on your CFFORM tags. Listing 1 provides examples of performing custom validation on CFINPUT fields of type text, radio and checkbox. These validation scripts can be used for both CFML and HTML form fields. In your JavaScript simply pass the name of the form field as you would for HTML fields.



About the Author

Selene Bainum is a senior software engineer for iXL. She has over four years of ColdFusion experience and has been a member of Team Allaire since December 1997.

selene@webtricks.com

FIGURE 1: A complex form

Listing 1: ExtendCFFORM.cfm

```
<HTML>
<HEAD>
<TITLE>Extending CFFORM</TITLE>
<SCRIPT LANGUAGE = "JavaScript">
// Main validation function.
// Variables needed: form name
// All validation functions will be called from
// this function
// The actual validation functions will be outside
// of this function, either above or below it
// If any of the variables used in these functions
// are ColdFusion variables, the functions must be
// within CFOUTPUT tags
function ExtendJS(FormName) {
    // Call the function to validate a date as being
    // before today's date.
    // Variables to pass: form name, field name,
    // type
    // The type parameter will have a value of
    // either 'date' or 'eurodate'
    if (!DateLessThan(FormName, 'DOB', 'date')) {
        alert("Your DOB cannot be after today.");
        return false;
    }
    // Call the RadioCheckBox function to
    // ensure one of a radio button series is
    // checked
    // Variables to pass: form name, field name,
    // number of items total, min required, max
    // required
    // * For radio fields, min and max will be 1
    // * The number of items total will most likely
    // be the RecordCount of the query used to
    // generate the options.
    if (!RadioCheckBox(FormName, 'Sex', 2, 1, 1)) {
        alert("You must select your sex.");
        return false;
    }
    // Call the RadioCheckBox function to
    // ensure one (or more) of a checkbox series is
    // checked
    // Variables to pass: form name, field name,
    // number of items total, min required, max
    // required
    // * The number of items total will most likely
    // be the RecordCount of the query used to
    // generate the options.
    if
        (!RadioCheckBox(FormName, 'Hobbies', 4, 2, 3))
    {
        alert("You must select 2 or 3 hobbies.");
        return false;
    }
    // Call the SingleSelectRequired function to
    // ensure an item in a select box is chosen
    // Variables to pass: form name, field name
    if (!SingleSelectRequired(FormName, 'Version')) {
        alert("You must a version of CF.");
        return false;
    }
    // Call the MultiSelectCheck function to ensure
    // one (or more) items in multi-select box are
    // chosen
    // Variables to pass: form name, field name
    if (!MultiSelectCheck(FormName, 'FavTags', 2, 6)) {
        alert("You must at least 2 tags.");
        return false;
    }
    // Call the TextAreaRequired function to ensure
    // a textbox field is filled out
    // Variables to pass: form name, field name
    if (!TextAreaRequired(FormName, 'Comments')) {
        alert("You must enter some comments.");
        return false;
    }
}

// Validation functions

// Function to ensure a text area field is filled
// out.
// Variables needed: form name, field name
function TextAreaRequired(Form, Field) {
    // determine the length of the value of the
```

```
// field.
// The eval function will evaluate the statement
// inside the parenthesis.
var length = eval("document." + Form + "." + Field
    + ".value.length");
// If the length of the value of the field is 0,
// return false.
if (length == 0) {
    return false;
} else {
    return true;
}
}

// Function to validate the number of items
// selected in a multi-select box
// Variables needed: form name, field name,
// minimum items selected,
// maximum items selected
function MultiSelectCheck(Form, Field, minReq, maxReq)
{
    // Set the selected var to a default of 0.
    var selected = 0;
    // Determine how many options are in the
    // multi-select box
    var length = eval("document." + Form + "." + Field
        + ".length");
    // Loop through the items in the multi-select
    // box
    for (i=0; i<length; i++) {
        // If an item is selected, increment the
        // selected var by one
        if (eval("document." + Form + "." + Field + "["
            + i + "].selected")) {
            selected++;
        }
    }
    // Determine if the number of items selected is
    // between the max and min values.
    // Return false if not, true if so
    if (selected < minReq || selected > maxReq) {
        return false;
    } else {
        return true;
    }
}

// Function to ensure that a value in a select box
// is selected
// Variables needed: form name, field name
function SingleSelectRequired(Form, Field) {
    // Determine the index of the item currently
    // selected
    var itemSelected = eval("document." + Form + "." +
        Field + ".selectedIndex");
    // If that index is 0, return false
    if (itemSelected == 0) {
        return false;
    } else {
        return true;
    }
}

// Function to validate radio and check boxes
// For radio, ensures that one radio button in a
// series is selected
// For checkbox, ensures that the number selected
// is in the min and under the max
// Variables needed: form name, field name, number
// of items total, minimum items checked, maximum
// items checked
// For radio buttons, the min and max will always
// be 1
function
    RadioCheckBox(Form, Field, length, minReq, maxReq)
{
    // Set the checked var to a default value of 0
    var checked = 0;
    // Loop through the number of items
    for (i=0; i<length; i++) {
        // If an item is checked, increment the
        // checked var
        if (eval("document." + Form + "." + Field + "["
            + i + "].checked")) {
            checked++;
        }
    }
}
```



LEVERAGING
BUSINESS
IN THE
**UNWIRED
WORLD**

CONFERENCE
December 3-5, 2000
Doubletree San Jose Hotel
San Jose, CA

**Early Bird
Registration:**
Register before
October 27th
and **SAVE**
\$200!

www.WIRELESSDEVCON2000.com

allaire

www.allaire.com

Developing Wireless Apps with ColdFusion Part 1

Setting the stage for growing acceptance

BY
CHARLES
AREHART



There's a growing buzz around the world of wireless Web applications, that is, Web sites accessible to phones and other wireless devices. You may have read that in ColdFusion 4.5 Allaire added support for something called WAP (Wireless Access Protocol).

It's the core of a powerful (and surprisingly easy) approach to creating applications for phones, pagers, two-way radios and more.

Perhaps you're wondering how all that works and whether you should jump on the bandwagon. How do you get started? How do you make it work in ColdFusion? What version of CF do you need? Are all wireless phones alike? What are the programming challenges of things such as browser detection, error handling, security and session management?

In this series of articles I hope to lay a foundation of principles and understanding to determine if and how you should go about developing wireless applications in ColdFusion. It's easy to do, but there are quite a few sources of confusion, so don't be surprised if your first attempts to look into it leave you wondering.

Much of the information in these articles is excerpted from my chapter on developing dynamic WAP applications using CF from the upcoming book, *Professional WAP Programming* (Wrox Press), which will already be out by the time you read this.

In this article, however, I'll lay some groundwork that I didn't get to address in that chapter (since mine was a later chapter in the book, and other writers addressed general WAP issues prior to it). I'll also share some of my own take on the state of affairs revolving around WAP application development.

First I want to address the question of if and how you should get started with WAP programming. I'll also introduce the challenges of converting an existing site to be



supported on wireless devices.

More important, I'll show you how to create and begin testing WAP applications even if you don't have a wireless phone so you can at least get started.

I'll also identify some resources for learning more on your own.

In the remaining articles in this series, I'll address the specifics of doing WAP development in ColdFusion and those challenging programming issues I mentioned above.

Building on Previous Articles

This article builds on two previous ones that appeared in *ColdFusion Developer's Journal*.

In the December issue (Vol. 1, issue 6) the illustrious Ben Forta wrote "No Strings Attached," a great introductory article on WAP and WAP programming. He laid out some of the foundation principles of WAP and showed some basic WML (Wireless Markup Language) code. You can find it online at www1.sys-con.com/coldfusion/archives/0106/forta/index.html.

In the April issue (Vol. 2, issue 4) Paul Elisii wrote an excellent article entitled "ColdFusion in the Palm of Your Hand" (www.sys-con.com/coldfusion/archives/0204/elisii/index.html). His focus was developing wireless applications for the Palm

computing platform (PalmPilots and related devices). As he indicated, the “Web clipping” approach he described is limited to those devices, and he mentioned WAP as a competing standard typically found in wireless phones.

I recommend that you read both as a precursor to this article to get a taste of the WML language *and* to tide you over until next month when I’ll begin showing some substantial examples and solutions.

Wireless Application Protocol

WAP-enabled phones are increasingly common; many if not most newly manufactured mobile phones now support the protocol.

The “wireless Web,” as many phone companies call it, will open a whole new world for accessing information on the Web in a timely and succinct manner (and, again, on more than just phones).

The best news of all (for readers of this magazine) is that you can use ColdFusion to create and drive such wireless Web sites, so it also opens up huge opportunities for those willing to invest a little time in learning about WAP and the WML language.

Created as an open protocol managed by the independent WAP Forum (www.wapforum.org), WAP is similar to – and in fact works in conjunction with – HTTP. The WML language is similar to HTML, but different enough so you really need to learn more than just the differences in syntax.

Developing for wireless devices really requires a new conception of what (and how much) to present to users and how to enable them to traverse the “site” you’re offering. Of course, the limited display interface is the most obvious challenge, but data entry is also extremely challenging on those simple keypads (see Figure 1).

Don’t let the minimal screen and features of the phone scare you away. With clever design, many powerful applications can leverage the ubiquity and flexibility – and indeed the minimalist interfaces – of these phones.

WML is easy to learn and should help wireless application development take off, much like Web page development did with the ease of

learning and developing in HTML.

Slow Uptake of WAP Phones

Still, there will inevitably be those who will point to articles and press releases foretelling the doom of WML, and to the slow uptake of WAP phones, especially in the U.S. They’re actually quite popular in Europe, Australia and Asia (in Japan there’s also another wireless protocol growing in popularity, called iMode).

WAP phones and wireless Web services aren’t yet taking the world by storm for a few reasons. I think it’s just a matter of time, but these problems are indeed a stumbling block to broader reception in the near term. Should you wait? I don’t think so. As I’ll explain here, much of the consternation comes from simple confusion about the new technology. However, it won’t take long for these problems to be overcome.

First, most phone service providers charge extra to access the wireless Web, but, fortunately, many are now letting it simply count against your talk-time minutes.

A considerable source of confusion (for users and information providers debating whether to “go wireless”) is the limited number of sites listed on the front of the phones by default (as shown in Figure 1). If you’re not listed there, how are people going to know about your site, let alone visit?

This is similar to the default portals that show up for users who install new copies of IE or Netscape. Novice users think “the Web” is only what’s listed on Microsoft’s MSN or Netscape’s NetCenter sites. But we know those sites are simply ones that have established relationships with Microsoft or Netscape, enabling them to be listed there.

While most users know (or soon learn) they can type in any Web address (URL) on their browsers, many wireless phone users face a greater challenge when doing so. First, they need to know they *can* type in any address (it’s usually buried in a menu system). Then they need to type it in laboriously using the phone’s limited keyboard (shortened domain name aliases are becoming popular for many well-known sites).

More important, users need to know that the sites they’re interest-



FIGURE 1: WAP phone

ed in (from broadly popular ones to those focused on some specific interest, or even an intranet) are available for wireless browsing. Many of the more popular sites are now creating wireless versions of their sites (often using the same URL but detecting and redirecting wireless visitors to a separate, WML-enabled section, as we’ll discuss later). Like the introduction of a new “traditional” Web site, promotion and marketing of a “wireless” site is critical.

Finally, many users (and, again, potential information providers) are under the false impression that the wireless Web is merely about presenting mini versions of existing Web sites. While it’s conceptually true, the wireless Web is actually about creating an entirely new class of applications suited to the smaller screen and, more important, leveraging the “always on, always at hand” nature of the phone.

Creating wireless apps is about rethinking what your users need, and either building modified versions of your existing applications to serve their unique needs or seeing the opportunity to create an entirely new application for wireless visitors.

In many ways this is like the early days of the Web – a lot of confusion, some hemming and hawing, and plenty of opportunity for early movers. Early adopters often get a bit bloodied, but in this case the significant similarity to HTML makes an investment of time and energy in WML seem very worthwhile, yet relatively inexpensive. It’s really easy to get started. So let’s do it.

Visiting (or Creating) a WML Site

Wireless Web sites (developed with WML) can't be viewed (yet) in normal HTML browsers. To do wireless development – or simply to visit some sites to get an idea of how things work – you don't need to have access to a wireless phone. If you do have access, great (though not all phones are created equal – more on that in a moment).

If you don't have a wireless phone, don't despair. Just head over to www.phone.com (the makers of the UPbrowser, which is at the heart of many WAP-enabled phones). They have a simulator you can install on your workstation that presents a life-like replica of a phone (see Figure 1, taken from the simulator) that allows you to enter the URL of any site, whether on your local workstation or on any site on the network (Internet or intranet).

To get the free simulator (what they call the UP.SDK or software developer kit) you'll need to register (again free) on their developer site. It's painless (and doesn't seem to lead to a lot of junk mail, etc.) and a link is clearly offered on their Web site.

The term *SDK* is a bit of a misnomer. It conjures up the notion of APIs you need to learn, installable libraries you need to compile and "distributions" you need to keep up with. In this case it's a simple, single-installation program that loads the simulator and a couple of related programs onto your workstation.

The SDK also installs some excellent documentation, including getting started, tutorial and reference guides for both WML and the phone (as well as WMLScript, discussed later).

The phone.com developer site has additional resources for learning more, including a technical library, a training program and even alternative "skins" to make the simulator look like your favorite phone.

Once you install and start the simulator, it'll load the page shown in Figure 1 (if you're connected to the Internet). From there you can browse the sites offered (which include some interesting how-to's, examples and links to real WAP-enabled sites).

You can also visit sites of your own choosing. On a real phone you'd have to find the menu com-

mand that allows you to enter the URL using the keypad (did I say how annoying that is?). Fortunately, in the simulator you can simply type in a URL with your keyboard using the top GO line of the simulator window (not shown in Figure 1, but obvious in the simulator). Again, you can enter any valid URL to open a WAP-enabled Web site, whether local or on the Internet. (I should add that users of real phones can usually bookmark a site address that they've "typed" in, saving them from frustration in future visits.)

Other WAP Phones and Simulators

While the phone.com simulator and the UPPhone browser included in many (if not most) wireless phones are very popular, they're not the only game in town. Sadly, like the challenges of different browsers in the early days of the Web (which still haunt us today), multiple WAP-enabled browsers are used to power phones and alternative simulators for those WAP browsers.

Nokia, a prominent wireless phone manufacturer, has their own embedded phone browser and simulator. While also based on WAP, it may process a page differently than a phone.com driven phone or simulator. Part of the problem is that the phone.com browser (in the phones and the simulator) allows for extensions to the WML language, just as Netscape extended HTML early on.

The formal specification (or DTD, Document Type Definition) for WML is laid out by the WAP Forum. If you rely solely on the phone.com simulator or a phone.com-driven phone for your testing and development, you run the risk of creating code that won't run in all phones. This is a reality. Many phones do embed the phone.com browser. However, it's up to you to decide what level of effort you want to expend on avoiding browser-specific WML and testing on multiple phones and simulators.

For early testing (rather than full-blown production rollout), it may be safe to let this issue slide while the industry sorts out these challenges. (Phone.com offers a 16-page white paper addressing this very issue of compatibility and standards integration. See www.phone.com/pub/IOTWP_0400.pdf.)


Next Issue, and Learning More in the Meantime

We're out of time and space. In the next issue I'll show you how to create WML applications in ColdFusion. It's really very easy. Remember that Ben Forta's December article shows the basics: the CFCONTENT tag and the special value needed for its "type" attribute, some simple WML code, issues such as case sensitivity of WML and more.

To learn more about WML and WAP, see the phone.com (or Nokia) documentation. Links to both are offered at a new developer area "reference desk" on the subject of WAP, available on the Allaire site at www.allaire.com/developer/Tech-nologyReference/wap.cfm.

Ben Forta recently took part in an interesting Q&A session on WAP and CF directions, available at www.allaire.com/handlers/index.cfm?id=15970&method=full.

In future articles I'll provide more substantial examples of WML development in ColdFusion. I'll also share information on some important items that will inevitably arise as you begin developing WML applications. I'll continue the discussion by identifying several tricks and traps to be aware of as you pursue this technology.

It's ever evolving, as is Allaire's support for it. It's an exciting time, and I hope you enjoy the ride! 

EDITOR'S NOTE

CFDJ readers who wish to explore issues regarding developing applications in the unwired world in more detail can go to www.sys-con.com/wireless where full details may be found of SYS-CON Media's newest magazine. The premier issue will include a feature article by Charles Arehart on "WML for HTML Developers."

ABOUT THE AUTHOR

Charles Arehart is a certified Allaire trainer and CTO of SystemManage, an Allaire partner. He contributes to several CF resources and is a frequent speaker at user groups throughout the country.

CAREHART@SYSTEMMANAGE.COM

Southfield, MI**Cold Fusion Web Developer**

Be a part of the entire Information Systems process at THE HOME DEPOT, the world's largest home improvement retailer.

The Special Order Center (S.O.C.) division is seeking a permanent full-time Cold Fusion Web Developer to maintain its existing home decor site while developing additional concepts for an expanded product line.

Applicants should have in depth knowledge of HTML, Cold Fusion, JavaScript, SQL, SQL Server and browsers. A working knowledge of Java, Perl, CGI, C++ and CORBA are a plus. Other requirements include a desire to learn, take on new responsibilities and grow along with the company. We are seeking highly motivated team players, with a can-do attitude, to work in a start-up like environment.

THE HOME DEPOT provides a balance between work and personal interests as seen through our **family friendly benefits** programs that include flexible hours, health insurance, 401(k), stock options, education reimbursement and a casual dress code. For immediate consideration, please contact:

Briana Motley
Home Depot Special Order Center
200 Galleria Offcentre 4th Floor
Southfield, MI 48034
Phone: 248-351-8700
Fax: 248-204-3734
briana_motley@homedepot.com



www.homedepot.com

Cambridge, MA**Cold Fusion Developers**

uclick is a pre-IPO company that is revolutionizing the Internet content market space. Soon we will be launching a number of major initiatives that will position us as a dominant industry player. We are on track to more than triple in size this year. We are looking for dynamic people with initiative who will help us maintain our excellent track record. uclick offers a (pre-IPO) stock option plan, competitive salary, 401K, vacation, casual attire, flexible hours, and growth opportunities in an exciting and fun environment. The Cambridge office is located next to the Alewife Train Station.

We are currently looking for Cold Fusion Developers and Sr. Cold Fusion Developers. Candidates should have 1+ years of Cold Fusion experience including knowledge of some advanced Cold Fusion features. Sr. Cold Fusion Developers should also have experience with scripted and non-scripted languages, knowledge of Internet protocols, and Oracle experience a plus. We are also looking for a Senior Architect and an Oracle DBA.

Please send resume and salary requirements to:
VP Engineering

uclick, LLC

125 Cambridgepark Drive, Cambridge, MA 02140
Email: lfillion@uclick.com
Phone: 617-868-0009 x203, Fax: 617-868-4344

GET YOUR OWN!

PowerBuilder
Enterprise Applications

JBuilder
Building Enterprise Portals

Developer's Journal
June 1999 Volume 1 Issue 3

COLD FUSION
ColdFusionJournal.com

XML: IT'S THE
XML-JOURNAL.COM

Call and Subscribe Today!
1-800-513-7111
www.sys-con.com

ADVERTISERS INDEX

ADVERTISER	URL	PH	PG
ABLECOMMERCE	WWW.AUCTIONBUILDER.COM	360.253.4142	2.
ABLECOMMERCE	WWW.ABLECOMMERCE.COM	360.253.4142	4
ALLAIRE	WWW.VUE.COM/ALLAIRE	877.460.8679	41
ALLAIRE	WWW.ALLAIRE.COM	888.939.2545	45,54
ALLAIRE DEVELOPER CONFERENCE	WWW.ALLAIRE.COM/CONFERENCE		31
CAREER OPPORTUNITIES		800.582.3089	59
COMPUTERWORK.COM	WWW.COMPUTERWORK.COM		35
CONCEPTWARE AG	WWW.CONCEPTWARE.COM	49.061.964.7320	21
CORDA TECHNOLOGIES	WWW.POPCHART.COM	888.763.0517	17
CYBERSMARTS	WWW.CYBERSMARTS.NET		62
DEVELOPERSNETWORK	WWW.DEVELOPERSNETWORK.COM	416.203.3690	15
DIGITALNATION	WWW.DEDICATEDSERVER.COM	877.624.7897	13
EKTRON	WWW.EKTRON.COM	603.594.0249	40
EPRISE	WWW.EPRISE.COM	508.661.5200	29
EVOLUTION B	WWW.EVOLUTIONB.COM/SYNERGYFORFREE	604.662.7551	63
FIG LEAF SOFTWARE	WWW.FIGLEAF.COM	202-797.5478	3
INTELIANT	WWW.INTELIANT.COM	800.815.5541	27
INTERLAND	WWW.INTERLAND.COM	800.419.1714	9
INTERMEDIA	WWW.INTERMEDIA.NET	650.424.9935	64
JDJ STORE	WWW.JDJSTORE.COM	888.303.JAVA	47
MACROMEDIA	WWW.MACROMEDIA.COM	415.252.2000	10,11
SAISOFT	WWW.SAISOFTONLINE.COM	860.793.6681	62
SITEHOSTING.NET	WWW.SITEHOSTING.NET	877.NTHOSTING	37
SYS-CON MEDIA INC.	WWW.SYS-CON.COM	800.513.7111	59
VIRTUALSEAPE	WWW.VIRTUALSEAPE.COM	212.460.8406	49
WIRELESS DEVCON	WWW.WIRELESSDEVCON2000.COM	212.251.0006	53
XML DEVCON 2000-2001	WWW.XMLDEVCON2000.COM	212.251.0006	61

Able Solutions

Enter the realm of browsable store building and administration – from your browser. Build “your_site.com” with secure Merchant Credit Card Processing. Maintain inventory, add discounts and specials to keep your customers coming back. Increase sales with cross selling and membership pricing.

11700 NE 95th Street, Suite 100, Vancouver, WA

www.ablecommerce.com • 360 253-4142

Catouzer

Catouzer develops web-based intranet and Customer Relationship Management software solutions. With Synergy 2.0, Catouzer continues its lead in providing secure web-based work environments. ColdFusion developers now have the most advanced framework to develop secure web-based projects.

www.catouzer.com • 604 662-7551

ComputerWork.com

ComputerWork.com is a premiere technical job site for computer professionals seeking employment in the IT/IS industry. ComputerWork.com will match your technical skills and career ambitions to our many employers looking to fill their jobs with specialists in computer related fields. You can submit your resume to a specific position on our job board or you can choose to submit your resume to our resume bank, which is accessed by nearly 400 companies nationwide. ComputerWork.com is the FASTEST way to your ideal career!

6620 Southpoint Drive South, Suite 600 Jacksonville, FL 32216

www.computerwork.com • 904-296-1993

Corda Technologies

Corda Technologies offers tools to build your own charts and graphs for internal reports, reports on your intranet and Internet sites and for many other applications where fast, high-quality graphs and charts are desirable. Corda also offers an Application Service Provider through PopChart.com which works with high-volume sports web sites to display sports statistics with exciting, interactive charts and graphs. PopChart!... an EXPLOSION of Possibilities!

1425 S. 550 East Orem, UT 84097

www.corda.com • 801-802-0800

DevelopersNetwork.com

Developers Network is the essential online business-to-business resource for new media technology and Internet business solutions. Our Resource, Business and Product channels combine elements of helpware and community in a business setting, successfully reaching those buyers developing and managing Internet strategies.

3007 Kingston Road Toronto, Ontario CANADA M1M 1P1

www.developersnetwork.com • 416-203-3610

www.cybersmarts.net

GoldFusion ASP Advertise PERL
100% Browser Based Admin Mail Domains FREE SSL
ODBC Log Files e-Commerce Solutions

Includes 10 Domains

cyberSmarts dot net

digitalNATION - a VERIO company

digitalNATION, VERIO's Advanced Hosting Division, is the world's leading provider of dedicated server hosting, with over 1,650 servers online today. dN's superior connected network and service abilities have earned dN a solid reputation as a first-choice provider of dedicated server solutions (Sun, Windows NT, Linux and Cobalt). digitalNATION has been providing online and network services for over six years. One of the first ISPs to pro-

vide dedicated servers running Microsoft Windows NT, the dN staff has unparalleled experience in this industry.

5515 Cherokee Ave, Alexandria, VA 22312-2309

www.dedicatedserver.com • 703 642-2800

Ektron

Ektron supports the next-generation needs of e-businesses by providing dynamic Web infrastructure solution tools designed for use by nontechnical staff. Ektron's flagship offering, eContentManager, gives staff members across an organization the hands-on ability to make real-time additions and updates to Web content without requiring knowledge of a programming language -- while still allowing for centralized administrative control and security. With competitive advantages such as ease-of-integration and drag & drop everything, Ektron is looking to provide these empowering products to customers, resellers and integrators.

5 Northern Blvd., Suite 6, Amherst, NH 03031

www.ektron.com • 603-594-0249

Eprise Corporation

At Eprise Corporation, we're dedicated to providing software, professional services and partnerships that make it easy to leverage the Web for more profitable and effective business operations. Our flagship product, Eprise Participant Server, incorporates leading technology to transform the dated, one-size-fits-all Web site into a strategic business asset that delivers timely and targeted communications. Simply put, Eprise and Eprise Participant Server empower business professionals to create, update, and target Web-based communications, regardless of their technical knowledge or skill. Contact us today to find out more about our products.

1671 Worcester Road, Framingham, MA 01701

www.eprise.com • 508-661-5200

FigLeaf Software

Fig Leaf Software specializes in developing turnkey web database applications and dynamic, data-driven websites. Our goal is to develop web-based client-server applications with functionality and interface design that are nearly indistinguishable from desktop software developed using traditional tools such as Visual Basic, Visual FoxPro, Delphi, or C. Above all, we want to bring maximum value to our clients at the minimum cost. The key to fulfilling this is ensuring our staff members are experts in their particular field. Our clients expect excellence, and we demand it of ourselves.

1400 16th St. NW, Suite 220, Washington, DC 20036

www.figleaf.com • 877.344.5323

Inteliant

Inteliant Corporation, a leading ColdFusion consulting firm, has an outstanding reputation for providing highly skilled developers for Internet, Intranet, Extranet, Software Development, or any ColdFusion application. Our national practice has emerged to meet the evolving needs of our clients by providing resources onsite or developing remotely. Our company provides the most cost effective service in the industry and we strive to add value to your projects by minimizing expenses whenever possible. Inteliant... "Delivering Intelligent Solutions

1150 Hancock Street, Suite 4, Quincy, MA 02169

www.inteliant.com • 800-815-5541

Interland

Interland, Inc., ranked the No. 1 Web Hosting Provider for small- to medium-sized businesses by Windows NT Magazine, Network Computing and PC Magazine, is a global leader in Web hosting solutions ranging from a basic Web site to a sophisticated e-commerce storefront.

Interland proudly features 24-hour, 7-day toll-free technical support and an advanced Administration Page. By deploying the best products, services, and support in the industry, Interland can build a Web presence that works for you. Speed. Reliability. Support. - Guaranteed.

101 Marietta Street, Second Floor, Atlanta, GA 30303

www.interland.com • 800-214-1460

Intermedia, Inc.

Our advanced virtual hosting packages (powered by Microsoft Windows NT and Internet Information Server 4.0) offer an environment supporting everything today's advanced Web developer or sophisticated client could ask for. Complete ODBC support is available on plans B and C. We support Microsoft Index Server on all hosting plans.

953 Industrial Avenue, Suite 121, Palo Alto, CA 94303

www.intermedia.net • 650 424-9935

Macromedia

New Macromedia UltraDev lets you create database-driven Web applications faster than ever before. It also allows you to create ASP, JavaServer Pages, and CFML applications in a single design environment. So whether you love morking directly with source code, or prefer to work visually, cut the time it takes to create employee directories, product catalogs, database search pages and more.

600 Townsend Street, San Francisco, CA 94103

www.macromedia.com • 415 252-2000

SaiSoft

As a recognized Allaire, Microsoft and IBM Solutions Provider, SaiSoft's Strategic focus is to become the most definitive Internet Architect, by building long lasting e-business development partnerships. With development operations in India & the UK, SaiSoft also undertakes off-shore consultancy projects where a 'four-step implementation' model is adopted to meet client needs satisfactorily.

446 East Street, Plainville, CT 06062

www.saisoftonline.com • 860-793-6681

Sitehosting.NET

Successful electronic commerce starts at SiteHosting.net; a division of Dynatek Infoworld, Inc., which provides total Web development services. We offer personal and efficient customer service with reliability at value prices. All our plans include access to SSL (Secure Socket Layer). We support ColdFusion, Active Server Pages, Real Audio/Video, Netshow Server, and more. Our hosting price starts at \$14.95/month.

13200 Crossroads Parkway North, Suite 360,

City of Industry, CA 91746

www.sitehosting.net • 877 684-6784

Virtualscape

Why host with Virtualscape? Nobody else on the Internet understands what it takes to host ColdFusion like we do. Virtualscape is the leader in advanced Web site hosting. From Fortune 500 extranets to e-commerce sites and more, developers recognize our speed, stability, reliability and technical support.

215 Park Avenue South, Suite 1905, New York, NY 10003

www.virtualscape.com • 212 460-8406

To place an ad in the
ColdFusion Marketplace
contact Robyn Forma at 201 802-3022

Optimized E-Business Solutions from Allaire, Intel
(Newton, MA) – Allaire is working closely with Intel Corporation to optimize Allaire ColdFusion for Intel Pentium III Xeon processors. The joint effort will deliver a new generation of large-volume, transaction-intensive Web applications.
www.allaire.com

ONCR Releases V4.0 of CFX_ONCR_CYBERCASH
(Metuchen, NJ) – Online Creations Inc. (ONCR), a leading developer of Internet software and services for companies building their businesses on the Web, has released CFX_ONCR_CyberCash 4.0, its custom tag for ColdFusion e-commerce applications.

The tag allows companies doing business on the Web to charge sales to a customer's credit card.

www.cfxoncrCybercash.com

Ektron Launches eWebEditPro 1.8
(Amherst, NH) – Ektron, Inc., has released the latest version of its Web content creation, editing and publishing tool. eWebEditPro version 1.8 features new Netscape compliance, international language support, an intuitive installation wizard and InterWoven TeamSite and IBM WebSphere sample code. Ektron also provides sample code for Allaire's ColdFusion and Spectra, JSP, PHP, Microsoft Active Server Pages and Vignette StoryServer.

www.ektron.com

Mercury Interactive Introduces Astra 4.0
(Sunnyvale, CA) – Mercury Interactive Corp.'s Astra LoadTest 4.0 and Astra QuickTest 4.0 Web testing tools are now available. The new features available in Astra 4.0 address

the expanding requirements for Web site testing while maintaining the product's ease of use. In addition, Mercury Interactive has introduced a Web-based training program for Astra LoadTest 4.0

designed to introduce and educate Web developers and QA teams on how to conduct successful load tests. The first Web-based training session is available for free. www.mercuryinteractive.com

SYS-CON Media to Publish Wireless Journal

(Montvale, NJ) – SYS-CON Media announced its newest publication, **Wireless Journal** (www.Wireless-Journal.com). WJs premier issue is scheduled for newsstands worldwide in September.

Crammed with product reviews, industry commentary and lively columns about unwired Life Beyond the PC, **WJournal** will be a mobile must-have. It will set the technology agenda, worldwide, for developers seeking to join or keep up with the fastest-growing area of software and hardware development since the arrival of the microprocessor.

SYS-CON Media, recently named the fastest-growing privately held publishing com-

pany in America by *Inc.* 500, is the world's leading publisher exclusively serving Internet technology markets. Publications currently are **ColdFusion Developer's Journal** (www.coldfusionjournal.com), **Java Developer's Journal** (www.javadevelopersjournal.com), **Java Buyer's Guide** (www.javabuyersguide.com), **XML Journal** (www.xml-journal.com), **XML Buyer's Guide** (www.xmlbuyersguide.com), **JBuilder Developer's Journal** (www.jbuilderjournal.com), **PowerBuilder Developer's Journal** (www.powerbuilderjournal.com) and **Tango Developer's Journal** (www.tangojournal.com).

ColdFusion Developer's Journal

Look What's Coming!

Attention Advertisers:

Don't Miss your chance to advertise in the
Allaire Developer Conference

Special Issue! Call Robyn Forma
201-802-3022

or email robyn@sys-con.com



www.cybersmarts.net

ColdFusion • ASP • ActiveState PERL

100% Browser Based Admin • Mail Domains • FREE SSL

ODBC • Log Files • e-Commerce Solutions

Includes 10 Domains

cyberSmarts
dot net



saisoft

www.saisoftonline.com

catouzer

intermedia

www.intermedia.net